CSE186 Winter 2024

Assignment 2

In this assignment you will write two simple JavaScript classes and tests to confirm they functions as expected. You will also gain some familiarity with well-known development tools for building, testing, documenting, and analyzing JavaScript programs outside the browser.

This assignment is worth 10% of your final grade.

Installation

Download and install node.js LTS for your operating system: https://nodejs.org/en/download/.

Do NOT install the “Current” version.

Note that node.js includes the Node Package Manager `npm`.

Once installed, open a terminal / console session and run the following command:

```
$ node --version
```

Which should return this version number:

```
V20.11.0
```

Now run the following command:

```
$ npm --version
```

Which should return one of these version numbers:

```
10.2.4
10.3.0
```

If you run into problems getting `node` and `npm` installed, come along to a TA or instructor office hours session as soon as possible so we can help you get up and running.

Setup

Download the starter code archive from Canvas and expand into an empty folder. I recommend, if you have not already done so, creating a folder for the class and individual folders beneath that for each assignment.

The starter code archive contains the following files:

```
README
package.json
rpnc.js
rpnc.test.js
templater.js
templater.test.js
```

You will modify `rpnc.js`, `templater.js`, `templater.test.js` and possibly `rpnc.test.js` but under no circumstances modify `package.json`.

To setup the node environment, ***navigate to the folder where you extracted the starter code*** and run the following command:

```
$ npm install
```

This will take some time to execute as `npm` downloads and installs all the packages required to build and test the assignment.

To execute the tests, run the following command:

```
$ npm test                    ( all tests )
$ npm test rpnc               ( just those for the basic requirement )
$ npm test template           ( just those for the advanced requirement )
```

To check your code quality against Google standards by running `eslint`, run the following command:

```
$ npm run lint                ( you'll need this for the stretch requirement )
```

Requirements

Basic:

Write a JavaScript Reverse Polish Notation (RPN) calculator supporting the following operators:

+	addition
-	subtraction
*	multiplication
/	division
^	power

Whilst you are welcome to implement additional operators, they will not be tested by the automated grading system. Take care not to waste time implementing operators that will gain you no credit.

Details of RPN can be found in many places on-line, but if you are unfamiliar the notation, a good place to start is its Wikipedia entry: https://en.wikipedia.org/wiki/Reverse_Polish_notation

Avoid the temptation to ask a code generator to write the calculator for you. Consult the lecture handouts, study the above link, work out what the parts of your code need to be and research how to do each one. For example, you will need to tokenize an RPN expression string into operators and operands. String tokenization is straight forward in JavaScript, so it won't take you long to find out how to do it.

Pseudo code for a stack-based RPC calculator is:

```
for each token in the postfix expression:
  if token is an operand:
    push token onto the stack
  else if token is an operator:
    operand_2 ← pop from the stack
    operand_1 ← pop from the stack
    result ← evaluate token with operand_1 and operand_2
    push result onto the stack
result ← pop from the stack
```

If you wish, you can add additional tests to `rpnc.test.js` but this is not required and may, in fact, indicate you have a suboptimal implementation.

Advanced Part I:

NOTE: Do not modify `template.test.js` yet, you'll do that later in Advanced Part II

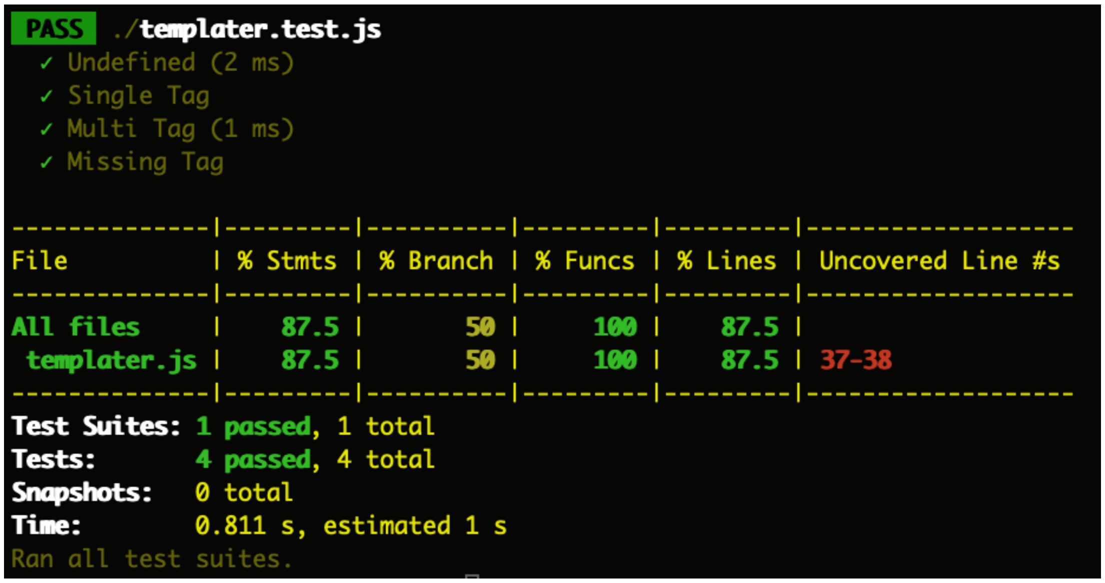
Implement a JavaScript class that when supplied with a template including tags enclosed in double curly braces (e.g. `{{sometag}}`) produces a string with the tags replaced by values found in a name-value-pair map i.e. a simple JavaScript object.

For example, after running the following code:

```
const t = new Templater('Hello {{tag}}');
let s = t.apply({tag: 'World'});
```

The variable `s` will have the value `'Hello World'`.

To get this working, complete the implementation of the `Templater` class in `templater.js` so it passes the tests provided in `templater.test.js`.
Once complete, running the tests (see "Setup" section) should produce output something like this:



```
 PASS  ./templater.test.js
  ✓ Undefined (2 ms)
  ✓ Single Tag
  ✓ Multi Tag (1 ms)
  ✓ Missing Tag

--------------|---------|----------|---------|---------|-------------------
File          | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
--------------|---------|----------|---------|---------|-------------------
All files     |    87.5 |       50 |     100 |    87.5 |
 templater.js |    87.5 |       50 |     100 |    87.5 | 37-38
--------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.811 s, estimated 1 s
Ran all test suites.
```

Note that the special cases of an undefined template or missing tag values in the map supplied to `apply()` have to handled correctly.

For missing tag values, any extra white space created by ***not*** replacing a tag must be removed. Note this is an unrealistic requirement but adds a little twist to the academic exercise. In a production templagter used in a Web App, the extra white space would not be removed.

For example, running the following code:

```
const t = new Templater('Mary {{had}} a {{little}} {{lamb}}');
let s = t.apply({had: 'had', lamb: 'lamb'});
```

Should give `s` the value `'Mary had a lamb'` with exactly one space between each word.

Advanced Part II:

Modify `template.js` to ensure the `Templater` class has the following features:

When the `strict` parameter to `apply()` is `true` and one or more tags are missing from `map`, an `Error` should be thrown. For example, when the following code is executed, an `Error` should be thrown by the second line:

```
let t = new Templater('Mary {{had}} a {{little}} {{lamb}}');
let s = t.apply({had: 'had', lamb: 'lamb'}, true));
```

In addition, if a tag in the template has whitespace in it, it should be ignored. For example, when the following code is executed (note the whitespace in the ‘`had`’ tag):

```
let t = new Templater('Mary {{had }} a {{little}} {{lamb}}');
let s = t.apply({had: 'had', little: 'little', lamb: 'lamb'}));
```

The variable `s` will have the value `'Mary a little lamb'`.

The following additional scenarios should be tested:

- There are no spaces between tags in the template
  `'Mary {{had}}{{little}}' => 'Mary hadlittle'`

- Tags appear more than once in the template
  `'Mary {{had}} {{had}}' => 'Mary had had'`

- Tags are separated by characters other than spaces in the template
  `'Mary {{had}}-{{little}}' => 'Mary had-little'`

We strongly suggest you consult the existing test code and the documentation at https://jestjs.io then modify `template.test.js` to include this new functionality, but you do not have to do that, you can test manually if you like until you get to the Stretch requirement.

Your code will be graded against known good tests so take care to ensure your implementation does what is expected of it.

Stretch:

Your code and tests have no lint errors or warnings, and your implementation exhibits 100% statement, branch, function, and line coverage.

Note that the intent is for you to tackle this requirement once Advanced has been met but failing part or all of Advanced does not mean you automatically fail this requirement. You must, however, implement automated tests for the Advanced requirement to pass Stretch.

What steps should I take to tackle this?

Review the lecture handouts, ask questions on Slack and consult any on-line resources you find useful. If you get stuck, come along to office hours to ask questions and get some help.

How much code will I need to write?

A model solution that satisfies all requirements adds approximately 30 to 40 lines of code to `rpnc.js`, 20 lines to `templater.js` and up to 60 lines to `templater.test.js`.

Grading scheme

The following aspects will be assessed:

1. (100%) **Does it work?**

a. Basic Requirement	(40%)
b. Advanced Requirement	(40%)
c. Stretch Requirement	(20%)

2. (-100%) **Did you give credit where credit is due?**

   a. Your submission is found to contain code segments copied from on-line resources and you failed to give clear and unambiguous credit to the original author(s) in your source code (-100%). You will also be subject to the university academic misconduct procedure as stated in the class academic integrity policy.

   b. Your submission is determined to be a copy of a past or present student's submission (-100%)

   c. Your submission is found to contain code segments copied from on-line resources that you did give a clear an unambiguous credit to in your source code, but the copied code constitutes too significant a percentage of your submission:

o < 25% copied code	No deduction
o 25% to 50% copied code	(-50%)
o > 50% copied code	(-100%)

What to submit

Run the following command to create the submission archive:

```
$ npm run zip
```

**** UPLOAD** `CSE186.Assignment2.Submission.zip` **TO THE CANVAS ASSIGNMENT AND SUBMIT ****