A Roman numeral represents an integer using letters. Examples are XVII to represent 17, MCMLIII for 1953, and MMMCCCIII for 3303. By contrast, ordinary numbers such as 17 or 1953 are called Arabic numerals. The following table shows the Arabic equivalent of all the single-letter Roman numerals:

M	1000	X	10
D	500	V	5
C	100	I	1
L	50		

When letters are strung together, the values of the letters are just added up, with the following exception. When a letter of smaller value is followed by a letter of larger value, the smaller value is subtracted from the larger value. For example, IV represents 5 - 1, or 4. And MCMXCV is interpreted as M + CM + XC + V, or 1000 + (1000 - 100) + (100 - 10) + 5, which is 1995. In standard Roman numerals, no more than three consecutive copies of the same letter are used. Following these rules, every number between 1 and 3999 can be represented as a Roman numeral made up of the following one- and two-letter combinations:

M	1000	XL	40
CM	900	X	10
D	500	IX	9
CD	400	V	5
C	100	IV	4
XC	90	I	1
L	50		

Create a library that takes a value, and returns a new object that can be used in the following way:

```
var romanNumber1 = new RomanNumber(‘XX’);
var romanNumber2 = new RomanNumber(40);

console.log(romanNumber1.toInt());  // => 20
console.log(romanNumber1.toString());  // => ‘XX’

console.log(romanNumber2.toInt());  // => 40
console.log(romanNumber2.toString());  // => ‘XL’
```

If the value passed:

- Is null or empty, it should throw a ‘value required’ exception error (e.g. ‘throw new Error(‘value required’);’)
- Is outside of 1 to 3999, it should throw an ‘invalid range’ exception error.
- Is invalid, it should throw an ‘invalid value’ exception error.

If the library is called as a function (i.e. without the new prefix), ensure you still pass back a new object.

Create the code using the TDD methodology, and test for the following values:

> null, ‘’, 0, 1, 3, 4, 5, ‘I’, ‘III’, ‘IIII’, ‘IV’, ‘V’, 1968, ‘1473’, 2999, 3000, 10000, ‘CDXXIX’, ‘CD1X’, ‘error’, ‘MCDLXXXII’, ‘MCMLXXX’, ‘MMMMCMXCIX’, ‘MMMMDMXCIX’

For the exercise, we are happy for the code and tests to be in the form of a single JS file. E.g.:

```
var    testChars = ['I', 'V', 'X', 'C'],

       isItTen = function(num) {
              return num.toLowerCase() === 'x'
       },

       isItFive = function(num) {
              return num.toLowerCase() === 'v'
       },

       testRoman = function(){
              testChars.forEach(function (str) {
                     console.log('Is %s 10? ', str, (isItTen(str) ? 'Yes' : 'No'));
                     console.log('Is %s 5? ', str, (isItFive(str) ? 'Yes' : 'No'));
              });
       };

testRoman();
```

In the tests, thrown exceptions should be caught and reported on without breaking the code.

This test should be done using NodeJS - string manipulation libraries may be used, but obviously none that contain functions that convert to or from roman numerals.

Create a public github repo when you start the test, with an initial commit with a start timestamp in a readme file, once you have finished, add an end timestamp, and commit the final changes.

You should commit and push your changes about every 10 minutes - and when a significant change/breakthrough happens.

As soon as you have created the repo, let us know so that we may follow the commits as they happen.