****

**Coding Kata: Roman Numerals**

Please complete the following coding exercise. When complete, upload all your code to a public file sharing site and email access to your BBC HR contact, as well as giving it ‘Open Permissions’ for others to access it freely.

In your email please also explain your approach, assumptions made, and any caveats to your solution. The purpose of this exercise is not just to solve the problem; we are interested in *how you approach* the problem too.

Things we are looking for:

- High quality code

- Test approach

- Problem solving

If you have any other code samples you would like to submit to give us a better understanding of your technical skills, please submit them at the same time.

**The problem**

In Javascript, or PHP, write a class that can convert Integers to Roman Numerals and Roman Numerals to Integers.

For example, see the following sample inputs and outputs:

1 = “I”

5 = “V”

10 = “X”

3999 = “MMMCMXCIX”

**Caveat**: Only support numbers between 1 and 3999

For an in-depth description of Roman Numerals, see [Wikipedia](http://en.wikipedia.org/wiki/Roman_numerals).

**Extension**

**B**uild a simple web interface to your converter. This may be as a UI that can be used in a web browser, or a web service API.

**SOLUTION**

The challenge was to provide a graphical user interface to convert integers into Roman numerals. The input was restricted between 1 and 3999 (requirement).

**The converter**

The main challenge with Roman numerals is that there is an extra number in the middle of two powers of ten (**V** between **I** and **X**, **L** between **X** and **C**, **D** between **C** and **M**) and the subtraction rule (**IV**, or **IX** for instance).

First, it is worth noting that writing Roman numerals is simply appending the conversion of every place value of the expanded notation of the number. For instance, 2345 as written in base 10 can also be written as 5\*100 + 4\*101 + 3\*102 + 2\*103 = 5 + 40 + 300 + 2000 and will be converted into a Roman numeral by appending the conversions of 1000, 200, 30 and 4 i.e. MCCXXXIV.

The proposed solution uses that property and calculates the expanded notation of the input number, by ascending order of place value. Then, a mapping function is provided to map the elements of the array and get their Roman numeral representation. Finally, the main function reverses the new array, appends all the elements and returns the result string.

To write the mapping function, I used the fact that for every power of ten there is a lower Roman numeral, a middle one and an upper one to work with.

|  |  |  |  |
| --- | --- | --- | --- |
| **Number** | **Place value** | | |
| **Ones** | **Tens** | **Hundreds** |
| Lower number | I | X | C |
| Middle number | V | L | D |
| Upper number | X | C | M |

No matter which place value we are working with, the rules are similar (and only scaled up to bigger intervals). For instance, building **IV** is the same as building **XL** or **CD**. Once the place value is known, the rules are straightforward.

|  |  |  |  |
| --- | --- | --- | --- |
| **Rules** | **Place value** | | |
| **Ones** | **Tens** | **Hundreds** |
| Repeat the lower number | [1, 4[ | [10, 40[ | [100, 400[ |
| Append lower number and middle number | [4, 5[ | [40, 50[ | [40, 500[ |
| Append middle number and repetition of lower numbers | ]5, 9[ | ]50, 90[ | ]500, 900[ |
| Append lower number and upper number | ]9, 10[ | ]90, 100[ | ]900, 1000] |

After applying those rules, as explained before, the final array containing the Roman numeral values is concatenated in one string.

**User interface**

The user interface has been developed using Bootstrap and AngularJS.

**Testing**

Two functions need to be tested, the first one is the function that provides the expanded notation of the input number and the function that outputs the actual Roman numeral.

The testing of the first function is straightforward and can be done with a few tests. Only basics tests (1, 10, 100 and 1000) need to be carried out and a more complex case decomposing 2345 (or any 4-digit number).

To test the second function, since we established that the Roman numeral representation of a number was the concatenation of Roman numeral representation of the thousands, the hundreds, the tens and the ones, we only need to test the following numbers:

|  |  |
| --- | --- |
| **Place name** | **Tested values** |
| ones | 1 2 3 4 5 6 7 8 9 |
| tens | 10 20 30 40 50 60 70 80 90 |
| hundreds | 100 200 300 400 500 600 700 800 900 |
| thousands | 1000 2000 3000 |

Testing only those cases is necessary and sufficient since any number between 1 and 3999 is a sum of some of those numbers. Therefore, we need to carry out only 30 tests. The tests have been carried out using the testing framework Mocha and linked on the main page.

**Source code**

The source code is available on GitHub at this address:

<https://github.com/gdoucet-fr/roman-numerals>

Once the repository is cloned, no packages need to be installed, and the interface can be accessed directly by opening **index.html**

A live demo is available at this address:

<https://gdoucet-fr.github.io/roman-numerals/>