# Technical Training Exercise: Roman Numerals

By Anthony McKale (Wizard without Portfolio)

The numeric system represented by Roman numerals originated in ancient Rome and remained the usual way of writing numbers throughout Europe well into the Late Middle Ages. Numbers in this system are represented by combinations of letters from the Latin alphabet. Roman numerals, as used today, are based on seven symbols:

Symbol I V X L C D M

Value 1 5 10 50 100 500 1,000

The use of Roman numerals continued long after the decline of the Roman Empire. From the 14th century on, Roman numerals began to be replaced in most contexts by the more convenient Hindu-Arabic numerals; however, this process was gradual, and the use of Roman numerals persists in some minor applications to this day.

<https://en.wikipedia.org/wiki/Roman_numerals>

There were certain rules that the numerals followed which should be observed.

1. The symbols 'I', 'X', 'C', and 'M' can be repeated at most 3 times in a row.
2. The symbols 'V', 'L', and 'D' can never be repeated.
3. The '1' symbols ('I', 'X', and 'C') can only be subtracted from the 2 next highest values ('IV' and 'IX', 'XL' and 'XC', 'CD' and 'CM').
4. Only one subtraction can be made per numeral ('XC' is allowed, 'XXC' is not).
5. The '5' symbols ('V', 'L', and 'D') can never be subtracted.

In this exercise you have to do the following tasks by yourself:

* Setup an Skeleton Aurelia Project
* Implement a Roman Numerals converter ES6 Class
* Plumb class into a Roman Numeral web component
* Style the web component using SCSS

# Exercise Rules

- Work **MUST** be your own, and you **MUST** work on your own

* You can Google anything you need to find out
* You can ask colleagues for guidance but you **MUST** not let them do the tasks on your behalf
* You **MUST** not use an off the shelf Roman Numeral converter library

- Attribution **MUST** be given for any code taken from online sources such as 'stack overflow', or any 3rd party libraries

- All code **MUST** be checked in to version control on a regular basis, after completing each logical step in your implementation

* All SVN commit messages **MUST** be prefixed with your tech training JIRA number
* You **MUST** not commit your *node\_modules / jspm\_packages* folder to SVN

- All code **SHOULD** be documented using jsdoc and explain any algorithms being used

- You will have 4 days to complete all 4 parts of this task (try to complete a minimum one of part per day)

- There are bonus tasks for each part, if you complete tasks early, you can attempt these

- At the end of each day your work in SVN will be created into a Crucible review and reviewed by a senior developer, they will give feedback via Crucible code review tool

- Good luck, hope you enjoy this Technical Training Exercise

Code review guidelines <https://github.com/thoughtbot/guides/tree/master/code-review>

# Part 1: Create Hello World "Roman Numerals Converter" Application

<http://aurelia.io/docs> <https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext>

1) Read the Rules on the previous page

2) Download and install the ESNext Aurelia Skeleton Project

*Hint:* [*https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext*](https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext) *Hint:* [*https://github.com/aurelia/skeleton-navigation/blob/master/skeleton-esnext/README.md*](https://github.com/aurelia/skeleton-navigation/blob/master/skeleton-esnext/README.md)

3) Run the tests in the Aurelia skeleton project to ensure it's working correctly

*Hint: 'npm test'*

4) Run the Aurelia skeleton project to ensure it's working correctly

*Hint: 'gulp watch’*

5) SVN ignore *node\_modules / jspm\_packages* folder, and any other you don't think should be getting committed (e.g. npmjs downloaded packages, compiled files etc)

*Hint:* [*https://tortoisesvn.net/docs/release/TortoiseSVN\_en/tsvn-dug-ignore.html*](https://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-dug-ignore.html)

6) Configure the package.json to have the project name "Roman Numerals Converter"

7) Save your workspace to SVN provided

## Part 1: Bonus Tasks

- Update anything else you think will tidy up your project

e.g. get rid of old of skeleton code, add a page title etc

## Part 1: If your struggling

Don’t Panic !

- If you’re having trouble unzipping the git repo and committing it to SVN, there’s a zipped esnext skeleton to get you started

xxx

# Part 2: Roman Numerals ES6 class (with Unit Tests)

<https://en.wikipedia.org/wiki/Roman_numerals>

1) Create a ES6 class called RomanNumeralsConverter.js with two exported functions 'toRomanNumeral(arabicNumber)' 'toArabicNumber(romanNumeral)'

*Hint:* [*https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext/src*](https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext/src) *Hint:* [*https://hacks.mozilla.org/2015/04/es6-in-depth-an-introduction/*](https://hacks.mozilla.org/2015/04/es6-in-depth-an-introduction/) *Hint:* [*https://developer.mozilla.org/en-US/docs/Learn*](https://developer.mozilla.org/en-US/docs/Learn)

2) Create an ES6 spec file

*Hint:* [*https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext/test/unit*](https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext/test/unit) *Hint:* [*https://jasmine.github.io/2.0/introduction.html*](https://jasmine.github.io/2.0/introduction.html)

3) Run the tests in the Aurelia skeleton project to ensure it's working correctly

*Hint: 'npm test'*

4) Implement and unit test the toArabicNumber(romanNumeral) function, this should parse valid a roman numeral and return a number

Minimum unit tests to implement

|  |
| --- |
| it('simple toArabicNumber (I) == 1', () => {  expect(toArabicNumber ('I')).toEqual(1);  }) |
| it('multi toArabicNumber (MMVIII) == 2008', () => {  expect(toArabicNumber ('MMVIII')).toEqual(2008);  }) |
| it('simple subtraction toArabicNumber (IV) == 4', () => {  expect(toArabicNumber ('IV')).toEqual(4);  }) |
| it('subtraction toArabicNumber (XC) == 90', () => {  expect(toArabicNumber ('XC')).toEqual(90);  }) |
| it('big toArabicNumber (MMMCMXCIX) == 3999', () => {  expect(toArabicNumber ('MMMCMXCIX')).toEqual(3999);  }) |

5) Implement and unit test the toRomanNumeral (arabicNumber) function, this should parse valid a number and return a roman numeral

Minimum unit tests to implement

|  |
| --- |
| it('simple toRomanNumeral (1) == I', () => {  expect(toRomanNumeral (1)).toEqual('I');  }) |
| it('multi toRomanNumeral (2008) == MMVIII', () => {  expect(toRomanNumeral (2008)).toEqual('MMVIII');  }) |
| it('subtraction toRomanNumeral (4) == IV', () => {  expect(toRomanNumeral (4)).toEqual('IV');  }) |
| it('subtraction toRomanNumeral (90) == 'XC'', () => {  expect(toRomanNumeral (90)).toEqual('XC');  }) |
| it('big toRomanNumeral (3999) == 'MMMCMXCIX'', () => {  expect(toRomanNumeral (3999)).toEqual('MMMCMXCIX');  }) |

6) Save your workspace to SVN

## Part 2: Bonus Tasks

- creating using js linting such as "eslint"

*Hint:* [*https://eslint.org/*](https://eslint.org/) */* [*https://github.com/adametry/gulp-eslint*](https://github.com/adametry/gulp-eslint)

- implementing Roman Numeral Zero

|  |
| --- |
| it('zero fractions toArabicNumber (nulla) == 0', () => {  expect(toArabicNumber ('nulla')).toEqual(0);  }) |
| it('zero fractions toRomanNumeral (0) == nulla', () => {  expect(toRomanNumeral (0)).toEqual('nulla');  }) |

*Hint:* [*https://en.wikipedia.org/wiki/Roman\_numerals#Zero*](https://en.wikipedia.org/wiki/Roman_numerals#Zero)

- implementing Roman Numeral Fractions

|  |
| --- |
| it('simple fractions toArabicNumber (MCDLIIS) == 1452.5', () => {  expect(toArabicNumber ('MCDLIIS')).toEqual(1452.5);  }) |
| it('simple fractions toRomanNumeral (1452.5) == MCDLIIS', () => {  expect(toRomanNumeral (1452.5)).toEqual('MCDLIIS');  }) |
| // note for peer review, is CCXXXIV.I or CCXXXIVS.S correct, interesting mind puzzle  it('complex fractions toArabicNumber (CCXXXIV.I) == 234 + (11/12)', () => {  expect(toArabicNumber ('CCXXXIV.I')).toEqual(234 + (11/12));  }) |
| // note for peer review, is CCXXXIV.I or CCXXXIVS.S correct, interesting mind puzzle  it('complex fractions toRomanNumeral (234 + (11/12)) == CCXXXIV.I', () => {  expect(toRomanNumeral (234 + (11/12))).toEqual('CCXXXIV.I');  }) |

*Hint:* [*https://en.wikipedia.org/wiki/Roman\_numerals#Fractions*](https://en.wikipedia.org/wiki/Roman_numerals#Fractions)

- if you really want to excel and have lots of time left, implement large Roman Numeral Apostrophus or Vinculum systems

*Hint:* [*https://en.wikipedia.org/wiki/Roman\_numerals#Large\_numbers*](https://en.wikipedia.org/wiki/Roman_numerals#Large_numbers)

## Part 2: If your struggling

Don’t Panic !

- remember Roman Numeral wikipedia page

[*https://en.wikipedia.org/wiki/Roman\_numerals*](https://en.wikipedia.org/wiki/Roman_numerals)

[*https://www.mathsisfun.com/roman-numerals.html*](https://www.mathsisfun.com/roman-numerals.html)

# Part 3: Create Aurelia "roman-numerals-converter" Web Component

1) create an Aurelia Web Component called roman-numerals-converter-wc.js and a html file roman-numerals-converter-wc.html

*Hint:* [*https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext/src*](https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext/src) */* [*http://aurelia.io/docs*](http://aurelia.io/docs)

2) Hook your new Aurelia Web Component into the Skeleton App

*Hint:* [*https://github.com/aurelia/skeleton-navigation/blob/master/skeleton-esnext/src/users.js*](https://github.com/aurelia/skeleton-navigation/blob/master/skeleton-esnext/src/users.js)

*Hint:* [*https://github.com/aurelia/skeleton-navigation/blob/master/skeleton-esnext/src/app.js*](https://github.com/aurelia/skeleton-navigation/blob/master/skeleton-esnext/src/app.js)

3) Create the html for a simple application, that displays the Roman Numeral for any number put into a text input

*Hint 1: hook your ES6 class RomanNumeralsConverter.js/toRomanNumeral from part2 into the Aurelia Web Component view model*

*Hint 2:* [*http://aurelia.io/docs/binding*](http://aurelia.io/docs/binding)

*Hint 3:* [*http://aurelia.io/docs/templating*](http://aurelia.io/docs/templating)

4) extend the html, with a new field that displays the number for any roman numeral put into a 2nd text input

*Hint: use your ES6 class RomanNumeralsConverter.js/toArabicNumber from part2 into the Aurelia Web Component view model*

## Part 3: Bonus Tasks

- creating / using view model unit tests

*Hint:* [*http://aurelia.io/docs/testing/components*](http://aurelia.io/docs/testing/components)

- creating / using view model e2e tests

*Hint:* [*http://aurelia.io/docs/testing/end-to-end*](http://aurelia.io/docs/testing/end-to-end)

- combine both text input fields, so you can update the text in one and it'll update the other text field

# **Part 4: Style Website using** **SCSS**

1) install using a node package module SCSS

(hint <http://sass-lang.com/documentation/file.SCSS_FOR_SASS_USERS.html>)

2) hook up SCSS compilation into your aurelia Gulp tasks

(hint <https://github.com/aurelia/skeleton-navigation/tree/master/skeleton-esnext/build/tasks>)

3) style your webapp using SCSS to be pretty

use your imagination

(example ideas)

- add an image or two

- show off, spin a logo 3d using css animation,

- add bootstrap 3 to project, use it for layout

4) have fun

5) Save your workspace to SVN

6) Inform code reviewer you’re finished the exercise

## Part 4: Bonus Tasks

- using SASS's features to improve readability and maintainability

(hint <http://sass-lang.com/documentation/file.SCSS_FOR_SASS_USERS.html>)

- creating / using css linting such as "stylelint"

(hint <https://stylelint.io/>)