PROGRAMMING Python Exercise

**TABLE OF CONTENTS**

[Introduction 1](#_Toc112248778)

[Overview 2](#_Toc112248779)

[BEFORE STARTING 2](#_Toc112248780)

[Task 2](#_Toc112248781)

[Part 1: Create "Hello World" bare bones repo 3](#_Toc112248782)

[Part 2: Roman Numerals Python File 4](#_Toc112248783)

[**Part 2: Bonus Points** 4](#_Toc112248784)

[Part 3: Python Linting 5](#_Toc112248785)

[**Part 3: Bonus Points** 5](#_Toc112248786)

[Part 4: Python Testing 6](#_Toc112248787)

[**Part 4: Bonus Tasks** 6](#_Toc112248788)

[Part 5: Flask Functions 8](#_Toc112248789)

[**Part 5: Bonus Points** 8](#_Toc112248790)

# Introduction

You are an important developer at a museum, and one of the professors requests a tool for converting Ancient roman numerals into modern Arabic (0-9) numbers.

Being awesome and epic you jump to the task.

Writing a simple cli application, then a full server that served the conversions to the wider world !

# Overview

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:

Symbols  
I = 1  
V = 5  
X = 10  
L = 50  
C = 100  
D = 500  
M = 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.

# BEFORE STARTING

* Setup your machine, using the Python developer guide
* Work through the flask tutorial  
  <https://flask.palletsprojects.com/en/2.0.x/tutorial/>

# Task

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

* Setup a Git Repo
* Implement a Roman Numerals converter python functions
* Style your code using a linter
* Unit test your code
* Plumb code into a Flask web server

## Part 1: Create "Hello World" bare bones repo

Setup a Git Repo

<https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

Steps:

* Read the Rules if you haven't
* create a repo in github
* create a hello world python file that prints out "Hello World!"
* document how to run file in README.md  
  *hint:* [*https://daringfireball.net/projects/markdown/basics*](https://daringfireball.net/projects/markdown/basics)
* ensure your .gitignore is correct  
  *hint:* [*https://github.com/github/gitignore*](https://github.com/github/gitignore)<https://github.com/github/gitignore/blob/master/>
* Save your workspace to GIT

## Part 2: Roman Numerals Python File

Implement a Roman Numerals converter python functions

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

Steps:

* Create a python file called roman\_numeral\_converter.py with two functions
  + 'to\_roman\_numeral(arabic\_number: int) -> str'
  + 'to\_arabic\_number(roman\_numeral: str) -> int'
* Implement these functions so they do their conversions
* document how to run file in README.md
* Save your workspace to GIT

### **Part 2: Bonus Points**

* use package like click to make it a command line application

<https://click.palletsprojects.com/en/7.x/>

./roman\_numeral\_converter.py --roman=IV# prints out 4 ./roman\_numeral\_converter.py --numeral=4# prints out IV

## Part 3: Python Linting

Style your code using a linter

Nice linting guide:  
<https://docs.python-guide.org/writing/style/>

Steps:

* Setup black to run over code
* Clean up code based on lint recommendations
* document how to run linting in README.md
* Save your workspace to GIT

### **Part 3: Bonus Points**

* use make and a makefile to have easy run reformat commands,  
  also document this in your README.md

make lint# prints out linting results  
make reformat# use black to automatically clean up code

* use git pre commit hooks to run linting automatically when committing work

<https://pre-commit.com/>

## Part 4: Python Testing

Unit test your code

Nice testing guide:  
<https://docs.pytest.org/en/stable/>

Steps:

* Implement unit tests for the to\_arabic\_number (roman\_\_numeral) function, this should parse valid a roman numeral and return a number
* ***Minimum*** unit tests to implement

def test\_to\_arabic\_number\_1\_rta():  
 *"""simple to\_arabic\_number (I) == 1"""* assert to\_arabic\_number("I") == 1  
  
  
def test\_to\_arabic\_number\_2008\_rta():  
 *"""multi to\_arabic\_number (MMVIII) == 2008"""* assert to\_arabic\_number("MMVIII") == 2008  
  
  
def test\_to\_arabic\_number\_4\_rta():  
 *"""simple subtraction to\_arabic\_number (IV) == 4"""* assert to\_arabic\_number("IV") == 4  
  
  
def test\_to\_arabic\_number\_90\_rta():  
 *"""subtraction to\_arabic\_number (XC) == 90"""* assert to\_arabic\_number("XC") == 90  
  
  
def test\_to\_arabic\_number\_3999\_rta():  
 *"""big to\_arabic\_number (MMMCMXCIX) == 3999"""* assert to\_arabic\_number("MMMCMXCIX") == 3999

* Implement unit tests the to\_roman\_numeral (arabic\_number) function, this should parse valid a number and return a roman numeral
* ***Minimum*** unit tests to implement

def test\_to\_roman\_numeral\_1\_atn():  
 *"""simple to\_roman\_numeral (1) == I"""* assert to\_roman\_numeral(1) == "I"  
  
  
def test\_to\_roman\_numeral\_2008\_atn():  
 *"""multi to\_roman\_numeral (2008) == MMVIII"""* assert to\_roman\_numeral(2008) == "MMVIII"  
  
  
def test\_to\_roman\_numeral\_4\_atn():  
 *"""simple subtraction to\_roman\_numeral (4) == IV"""* assert to\_roman\_numeral(4) == "IV"  
  
  
def test\_to\_roman\_numeral\_90\_atn():  
 *"""subtraction to\_roman\_numeral (90) == XC"""* assert to\_roman\_numeral(90) == "XC"  
  
  
def test\_to\_roman\_numeral\_3999\_atn():  
 *"""big to\_roman\_numeral (3999) == MMMCMXCIX"""* assert to\_roman\_numeral(3999) == "MMMCMXCIX"

* document how to run tests in README.md
* Save your workspace to GIT

### **Part 4: Bonus Tasks**

* use git pre commit hooks to run testing automatically when committing work

<https://pre-commit.com/>

* implementing Roman Numeral Zero

zero fractions to\_arabic\_number (nulla) == 0  
zero fractions to\_roman\_numeral (0) == nulla

Hint: <https://en.wikipedia.org/wiki/Roman_numerals#Zero>

* implementing Roman Numeral Fractions

simple fractions to\_arabic\_number (MCDLIIS) == 1452.5  
// note for peer review, is CCXXXIV.I or CCXXXIVS.S correct, interesting mind puzzle'complex fractions to\_arabic\_number (CCXXXIV.I) == 234 + (11/12)'  
// note for peer review, is CCXXXIV.I or CCXXXIVS.S correct, interesting mind puzzle  
complex fractions to\_roman\_numeral (234 + (11/12)) == CCXXXIV.I'

Hint: <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>

## Part 5: Flask Functions

Plumb code into a REST endpoint and deploy to local server runnning on Flask

Steps:

* Convert your existing functions into flask format   
  (you can import existing function if easier /api/to-romain)
* Create webpage in Flask server to host some javascript to call the rest endpoint you created + css
* Use Make to run FLask server
* Unit test and integration test your Flask Server
* document how to run deploy and run in README.md
* Save your workspace to GIT

### **Part 5: Bonus Points**

* implement web frontend to flask endpoints