

**Homework #3****Due: 10/11**

This homework will test your ability to make use of Python's objects and magic methods. Think before you code: A bad design will waste your time and lead to buggy and hard-to-test code.

*Reminder: This assignment is to be your own work. You are not to “borrow” code from any source apart from the textbook.*

1. [ 100 points ] Implement a class **Roman** in a module `roman.py` that adds Roman numeral functionality to Python. The basic digits are:

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

In addition, a **bar could be put** over or parentheses put around any of these except I that would multiply it by 1000. Using the latter, we have:

numeral	decimal equivalent
(V)	5000
(X)	10000
(L)	50000
(C)	100000
(D)	500000
(M)	1000000

There are specific rules for how to turn any positive integer up to several million into roman numerals. For example, 37 is XXXVII and 99 is XCIX. If you don't remember these rules, the Wikipedia article “Roman Numerals” is a good review.

**Roman** should implement the following operations:

`x + y`   `x - y`   `x * y`   `x / y`  
`x // y`   `x ** y`   `x == y`   `x != y`  
`x < y`   `x <= y`   `x >= y`   `x > y`  
`-x`

Observe these additional constraints:

- The result of any unary arithmetic operation involving a **Roman** should be a **Roman**.
- The result of any binary arithmetic operation involving a **Roman** and an **int** should be a **Roman**.

- (Comparison operations still result in `bools`, of course.)
- Roman mathematicians did not have negative numbers, but `Roman` will indicate negative values with a `-` prefix.
- Roman mathematicians did not have a zero, but `Roman` will use `'N'` to stand for *nulla* (“nothing”). Note that `N` is *never* a placeholder: It is only used to indicate a value of 0.
- Floor (`//`) division returns a `Roman`, ignoring the remainder (as it should).
- True (`/`) division returns a `tuple` of two `Romans` “(*quotient*, *remainder*)”.
- Reuse magic methods wherever possible. E.g., `__radd__()` should call `__add__()`
- Values that are too big in absolute value to represent (let’s make it 2000000 or more for convenience), should raise a `ValueError` exception.
- `Roman()` should accept a mandatory `int` argument.

Here is an interactive example of the working module:

```
>>> from roman import Roman
>>> III = Roman(3)
>>> VII = Roman(7)
>>> print(III)
III
>>> III + VII
Roman(10)
>>> print(III + VII)
X
>>> print(VII + 2)
IX
```

Within the module, instantiate all roman numerals up to and including 1000 (i.e. `M`) as objects so that this (for instance) works:

```
>>> from roman import *
>>> III*XI + CM*II
Roman(1833)
```

There’s an easy way to do this (hint: `globals()`) and a hard way. Credit will not be given for the hard way.

[ +10 pts. extra credit ] Make the constructor `Roman()` also accept a legal roman numeral (as a `str`) and convert it for use internally. (Any illegal roman numeral string should raise an exception of your own design.) Also make this the default display format for the `__repr__()` method. This would make the line in the above example:

```
>>> III + VII
Roman('X')
```