Roman Number demonstration class.

This demonstration program defines and tests a programmer created object class.

romanclass.py

A “Roman” object is stored in the computer as a binary integer, but is displayed in Roman numerals. (In technical terms, it is a subset of the built-in class **int** with a _str_() method which converts to a Roman numeral string.) Roman objects act very much like the built-in Decimal objects in Python, they can be added, subtracted, multiplied, or divided and the result will be another object of the same class.

So a programmer can say:

```
import romanclass as roman
two = roman.Roman(2)
five = roman.Roman('V')
print two+five
```

and the computer will print:

```
VII
```

[Note: there was already a module called “roman.py” written by Mark Pilgrim and found in his book

[Dive Into Python, which is why this module is named “romanclass” to avoid conflict.

[the methods .toRoman() and .fromRoman in both modules will give compatible results

[within the restrictions of Mark's algorithm. (0 < n < 3999)

[>>> assert roman.toRoman(123) == str(romanclass.toRoman(123))

[>>> assert roman.fromRoman('IV') == int(romanclass.fromRoman('IV'))

CaeserCalc.py

The program “CaesarCalc.py” is a simple console mode calculator which tests the module by accepting a typed line with two numbers (Roman or Arabic) separated by '+' '-' '*' or '/'. It will interpret those numbers as Roman and calculate and print the result.

If you try to create a Roman number larger than 3999, your computer will probably not be able to print the result, because Roman digits for 5000 and larger cannot be represented in ASCII and must be output in unicode characters. Fonts which define all of the Roman numerals (especially 50,000 and 100,000) are very rare. [The “Code2000” font does include them.] If you attempt to print roman.Roman('MMMCMXCIX') + 1, it will emit the unicode for:

5,000 \u2181

MↃ

which is how you write 4000 (5000 – 1000). Here are the others:

10,000 \u2182

50,000 \u2187

100,000 \u2188

Other unicode Roman numeral characters exist, and you may send any numeric-valued character to roman.Roman() which will use the value defined by the unicode standard.

[Note: as of 2009-6-30 the codes for 50,000 and 100,000 may not work for input due to an error in Python's unicodedata module. (See issue 6383) The fix is included in Python 2.7. IronPython, as of version 2.6 has not implemented the unicodedata module at all, so cannot use any unicode characters for input.]

Installation:

If your computer does not have a copy of Python installed, download it from http://python.org/ and install it.

Copy (or extract) the files from romanclass.zip into a folder of your choice.

Change directory to that folder and execute the command: “python setup.py install”.

Copy CaesarCalc.py to a folder of your choice.

This module has been tested using CPython versions 2.6.4, 2.7, and 3.1, and IronPython versions 2.6 and 2.7.

If you run romanclass.py (or call the .test() method), it will run and print a self-check.

Interface:

`Roman()` will create an new instance of a Roman object. The argument may be either a string of Roman Digits, or any input (string or numeric) which is accepted by the int() built in function. In other words, the following calls are equivalent:

```
Roman('MCMLXVI')
Roman('mdccccxxxxxxvj')
Roman('1966')
Roman(1966)
Roman(1966.4)
```

Notice that the input routine for Roman numerals makes no requirement that the number be normalized according to modern standards. It will attempt to render any jumbled combination of suitable characters.

`Class methods:`

`fromRoman()`

Accepts a string of Latin numeric characters [IVXLDCM] and / or unicode characters which have a numeric value. (That is, where the value of unicodedata.numeric() is defined.) And attempts to parse them using a simple rule for the formation of Roman numbers. A blank input or the strings 'N' or 'Nulla' will be stored as Zero. Returns an integer number.

Values less than zero or greater than 599,999 will result in an OutOfRangeError exception. Invalid characters in the input stream will give an InvalidRomanNumeralError exception. Both are subtypes of ValueError.

`ToRoman()`

Accepts an integer and formats a unicode string as a normalized Roman number. The usual ASCII characters will be used for values in the range 1 ... 3999. Zero will be rendered as 'Nulla' which is the Latin word for 'nothing'. If the value is greater than 3999, unicode Roman Number characters will be used for the high order digits.

Values less than zero or greater than 599,999 will give an OutOfRangeError exception.

`toUnicodeRoman()`

Accepts an integer and formats a unicode string as a normalized Roman number. Zero will be rendered as 'Nulla'. Unicode Roman Number characters will be used for all digits. If the value is between 1 and 12 a single unicode character with the appropriate value will be returned. This feature could be used, for example, to generate label values for a clock face. Values outside that range will be rendered as strings of one or more unicode characters in the usual manner.

Values less than zero or greater than 599,999 will give an OutOfRangeError exception.

Mathematical Operations:

Addition, subtraction, multiplication and floor division are defined for the Class. The result will be the type of the left hand operand. Thus:

two = Roman(2)

print two + 2

IV

print 2 + two

4