```
# 1  py.test version of test.

import roman
from py.test import raises

known_values = """
1 I
2 II
3 III
4 IV
5 V
6 VI
7 VII
8 VIII
9 IX
10 X
50 L
100 C
500 D
1000 M
31 XXXI
148 CXLVIII
294 CCXCIV
312 CCCXII
421 CDXXI
528 DXXVIII
621 DCXXI
782 DCCLXXXII
870 DCCCLXX
941 CMXLI
1043 MXLIII
1110 MCX
1226 MCCXXVI
1301 MCCCI
1485 MCDLXXXV
1509 MDIX
1607 MDCVII
1754 MDCCLIV
1832 MDCCCXXXII
1993 MCMXCIII
2074 MMLXXIV
2152 MMCLII
2212 MMCCXII
2343 MMCCCXLIII
2499 MMCDXCIX
2574 MMDLXXIV
2646 MMDCXLVI
2723 MMDCCXXIII
2892 MMDCCCXCII
2975 MMCMLXXV
3051 MMMLI
3185 MMMCLXXXV
3250 MMMCCL
3313 MMMCCCXIII
3408 MMMCDVIII
3501 MMMDI
3610 MMMDCX
3743 MMMDCCXLIII
3844 MMMDCCCXLIV
3888 MMMDCCCLXXXVIII
3940 MMMCMX
3999 MMMCMXCIX"""
known_values = [tuple(line.split())
                for line in known_values.splitlines()
                if line]
def param(param_list, expand=True):
    def make_param_func(func):
        def yielder():
            for items in param_list:
                if not isinstance(items, tuple):
                    all = (func, items)
                else:
                    all = (func,) + items
                yield all
        if expand:
            return yielder
        else:
            def runner():
                for runner in yielder():
                    runner[0](*runner[1:])
            return runner
    return make_param_func
```

```
# 2  py.test version of tests.

@param(known_values)
def testToRomanKnownValues(integer, numeral):
    """toRoman should give known result with known input"""
    print "Trying %s -> %s" % (integer, numeral)
    assert numeral == roman.toRoman(integer)

@param(known_values)
def testFromRomanKnownValues(integer, numeral):
    """fromRoman should give known result with known input"""
    assert roman.fromRoman(numeral) == integer

def testTooLarge():
    """toRoman should fail with large input"""
    raises(roman.OutOfRangeError, roman.toRoman, 5000)

def testZero():
    """toRoman should fail with 0 input"""
    raises(roman.OutOfRangeError, roman.toRoman, 0)

def testNegative():
    """toRoman should fail with negative input"""
    raises(roman.OutOfRangeError, roman.toRoman, -1)

def testNonInteger():
    """toRoman should fail with non-integer input"""
    raises(roman.NotIntegerError, roman.toRoman, 0.5)

@param(('MMMM', 'DD', 'CCCC', 'LL', 'XXXX', 'VV', 'IIII'))
def testTooManyRepeatedNumerals(s):
    """fromRoman should fail with too many repeated numerals"""
    raises(roman.InvalidRomanNumeralError, roman.fromRoman, s)

@param(('CMCM', 'CDCD', 'XCXC', 'XLXL', 'IXIX', 'IVIV'))
def testRepeatedPairs(s):
    """fromRoman should fail with repeated pairs of numerals"""
    raises(roman.InvalidRomanNumeralError, roman.fromRoman, s)

@param(('IIMXCC', 'VX', 'DCM', 'CMM', 'IXIV',
        'MCMC', 'XCX', 'IVI', 'LM', 'LD', 'LC'))
def testMalformedAntecedent(s):
    """fromRoman should fail with malformed antecedents"""
    raises(roman.InvalidRomanNumeralError, roman.fromRoman, s)

@param(range(1, 4000), False)
def testSanity(integer):
    """fromRoman(toRoman(n))==n for all n"""
    numeral = roman.toRoman(integer)
    result = roman.fromRoman(numeral)
    assert integer == result

@param(range(1, 4000), False)
def testToRomanCase(integer):
    """toRoman should always return uppercase"""
    numeral = roman.toRoman(integer)
    assert numeral == numeral.upper()

@param(range(1, 4000), False)
def testFromRomanCase(integer):
    """fromRoman should only accept uppercase input"""
    numeral = roman.toRoman(integer)
    roman.fromRoman(numeral.upper())
    raises(roman.InvalidRomanNumeralError,
           roman.fromRoman, numeral.lower())
```