And One More Thing...

There was one more functional requirement for converting numbers to Roman numerals: dealing with non-integers.

```
import roman3
print (roman3.to_roman(0.5))  #①
#''

print (roman3.to_roman(1.0))  #②
#'I'
```

① Oh, that's bad.

② Oh, that's even worse. Both of these cases should raise an exception. Instead, they give bogus results.

Testing for non-integers is not difficult. First, define a `NotIntegerError` exception.

```
# roman4.py
class OutOfRangeError(ValueError): pass
class NotIntegerError(ValueError): pass
```

Next, write a test case that checks for the `NotIntegerError` exception.

```
import unittest

class ToRomanBadInput(unittest.TestCase):
    #.
    #.
    #.
    def test_non_integer(self):
        '''to_roman should fail with non-integer input'''
        self.assertRaises(roman4.NotIntegerError, roman4.to_roman, 0.5)
```

Now check that the test fails properly.

```
you@localhost:~/diveintopython3/examples$ python3 romantest4.py -v
test_to_roman_known_values (__main__.KnownValues)
to_roman should give known result with known input ... ok
test_negative (__main__.ToRomanBadInput)
to_roman should fail with negative input ... ok
test_non_integer (__main__.ToRomanBadInput)
to_roman should fail with non-integer input ... FAIL
test_too_large (__main__.ToRomanBadInput)
to_roman should fail with large input ... ok
test_zero (__main__.ToRomanBadInput)
to_roman should fail with 0 input ... ok

======================================================================
FAIL: to_roman should fail with non-integer input
----------------------------------------------------------------------
Traceback (most recent call last):
  File "romantest4.py", line 90, in test_non_integer
    self.assertRaises(roman4.NotIntegerError, roman4.to_roman, 0.5)
AssertionError: NotIntegerError not raised by to_roman

----------------------------------------------------------------------
Ran 5 tests in 0.000s

FAILED (failures=1)
```

Write the code that makes the test pass.

```
def to_roman(n):
    '''convert integer to Roman numeral'''
    if not (0 < n < 4000):
        raise OutOfRangeError('number out of range (must be 1..3999)')
    if not isinstance(n, int):                                      #①
        raise NotIntegerError('non-integers can not be converted')  #②

    result = ''
    for numeral, integer in roman_numeral_map:
        while n >= integer:
            result += numeral
            n -= integer
    return result
```

① The `built-in isinstance()` function tests whether a variable is a particular type (or, technically, any descendant type).

② If the argument `n` is not an `int`, raise our newly minted `NotIntegerError` exception.

Finally, check that the code does indeed make the test pass.

```
you@localhost:~/diveintopython3/examples$ python3 romantest4.py -v
```

```
test_to_roman_known_values (__main__.KnownValues)
to_roman should give known result with known input ... ok
test_negative (__main__.ToRomanBadInput)
to_roman should fail with negative input ... ok
test_non_integer (__main__.ToRomanBadInput)
to_roman should fail with non-integer input ... ok
test_too_large (__main__.ToRomanBadInput)
to_roman should fail with large input ... ok
test_zero (__main__.ToRomanBadInput)
to_roman should fail with 0 input ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
```

The `to_roman()` function passes all of its tests, and I can't think of any more tests, so it's time to move on to `from_roman()`.