Recap

- Validating, Verifying, Testing, Debugging
- Three types of Tests
- Two testing methodologies
- Why test boundaries?
if type(input) != type(1):
    raise TypeError, "expected integer, got %s" % type(input)
if not 0 < input < 4000:
    raise ValueError, "Argument must be between 1 and 3999"
ints = (1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1)
nums = ('M', 'CM', 'D', 'CD', 'C', 'XC', 'L', 'XL', 'X', 'IX', 'V', 'IV', 'I')
result = ""
for i in range(len(ints)):
    count = int(input / ints[i])
    result += nums[i] * count
    input -= ints[i] * count
return result
■ Testing should NOT be viewed as a separate phase.
■ Testing should be a continuous process, to be done at the same time as development.
Automated Testing

- Automated Testing is achieved by running tests using software and comparing results to predetermined values.
- Unit Tests tools are often used to achieve this goal.
- Automated testing can be used in different ways:
  - By a programmer to verify his code
  - At a specific time, to determine the correctness of the code in a repository.
  - When a programmer checks in his code, to determine the correctness of the addition.
Adding a new feature can sometimes have unforeseen effect on existing code.

To detect this problems, we should always run both old and new tests.
  - Note: running the full suite of time can be very time consuming (i.e. hours).

This is called regression testing.
Unit Testing

- Unit Testing is writing code that tests code that you haven't written yet.
- The keyword, unit, refers to the smallest piece of code that can be tested.
  - In OO language, this is a class.
- Unit testing is not a replacement for higher-level functional or system testing, but it is important in all phases of development:
- Unit testing started as a framework for testing SmallTalk code.
- It evolved into JUnit, and latter, into PyUnit.
Before writing code, it forces you to detail your requirements in a useful fashion.

While writing code, it keeps you from over-coding. When all the test cases pass, the function is complete.

When refactoring code, it assures you that the new version behaves the same way as the old version.

When writing code in a team, it increases confidence that the code you're about to commit isn't going to break other peoples' code, because you can run their unit tests first.
Components of a Unit Test

- **test fixture**
  - A test fixture represents the preparation needed to perform one or more tests, and any associate cleanup actions. This may involve, for example, creating temporary or proxy databases, directories, or starting a server process.

- **test case**
  - A test case is the smallest unit of testing. It checks for a specific response to a particular set of inputs. unittest provides a base class, TestCase, which may be used to create new test cases.

- **test suite**
  - A test suite is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.

- **test runner**
  - A test runner is a component which orchestrates the execution of tests and provides the outcome to the user. The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests.
- The test case and test fixture concepts are supported through the TestCase class.
- When building test fixtures using TestCase, the setUp() and tearDown() methods can be overridden to provide initialization and cleanup for the fixture.
- Each instance of the TestCase will only be used to run a single test method, so a new fixture is created for each test.
Test suites are implemented by the TestSuite class.

- This class allows individual tests and test suites to be aggregated; when the suite is executed, all tests added directly to the suite and in `child` test suites are run.

A test runner is an object that provides a single method, run(), which accepts a TestCase or TestSuite object as a parameter, and returns a result object.

- The class TestResult is provided for use as the result object.
import random
import unittest

class TestSequenceFunctions(unittest.TestCase):
    def setUp(self):
        self.seq = range(10)

    def testshuffle(self):
        # make sure the shuffled sequence does not lose any elements
        random.shuffle(self.seq)
        self.seq.sort()
        self.assertEqual(self.seq, range(10))

    def testchoice(self):
        element = random.choice(self.seq)
        self.assert_(element in self.seq)

if __name__ == '__main__':
    unittest.main()
testchoice (__main__.TestSequenceFunctions) ... ok
testsample (__main__.TestSequenceFunctions) ... ok
testshuffle (__main__.TestSequenceFunctions) ... ok

Ran 3 tests in 0.110s

OK
Continue Reading on PyUnit

- Python Library Ref

- Dive Into Python
  - http://www.diveintopython.org/unit_testing/index.html

- PyUnit
  - http://pyunit.sourceforge.net/
Mutable vs Immutable Objects

- An Immutable object is an object that is created once and is never changed.
  - String, Long, etc.
  - Two Immutable objects are considered the same if they have the same state.

- A Mutable object is an object who's state can change.
  - Vector, Array, etc.
  - Two different Mutable objects are never considered the same (different identity).