

Adopting standard testing framework

09/24/09

About a testing framework

- Roles of testing framework
 - Support test automation
 - Share setup and shutdown codes for tests
 - Aggregate individual tests
 - Separate tests from the reporting framework.
- Using standard framework, we can expect to:
 - Leverage open source libraries.
 - Concentrate on test cases.
 - Write tests in a standardized way.
 - Get contribution from other parties.

Testing framework candidate 1: PyUnit

- A standard unit testing framework module for Python.
- Part of Python's standard library (as the unittest module).
- Provides classes that make it easy to support qualities for a set of tests.

Testing framework candidate 2: py.test

- Has lighter-weight syntax for writing tests than PyUnit.
- Discovers tests automatically
by looking at specified directories and its files
- Supports many testing methods conventionally used in the Python community.
- Runs traditional unittest.py, doctest.py
- Supports xUnit style setup and nose specific setups and test suites.
- Integrates coverage testing with figleaf or Javascript unit- and functional testing.

Testing framework candidate 3: nose

- Has lighter-weight syntax for writing tests than PyUnit.
- Runs Class and functions matching regular expression
- nosetests and py.test share basic philosophy and features.
- Provides an alternate test discovery and running process for unittest (mimic the behavior of py.test without resorting to too much magic)
- Inspired mainly by py.test, but formerly was not all that easy to install, and is not based on unittest.
- Test suites written for use with nose should work equally well with py.test, and vice versa, except for the differences in output capture and command line arguments for the respective tools.