UnitTest Framework - Py.test Module

It was in 2004 that Holger Krekel renamed his **std** package, whose name was often confused with that of the Standard Library that ships with Python, to the (only slightly less confusing) name 'py.' Though the package contains several sub-packages, it is now known almost entirely for its py.test framework.

The py.test framework has set up a new standard for Python testing, and has become very popular with many developers today. The elegant and Pythonic idioms it introduced for test writing have made it possible for test suites to be written in a far more compact style.

py.test is a no-boilerplate alternative to Python's standard unittest module. Despite being a fully-featured and extensible test tool, it boasts of a simple syntax. Creating a test suite is as easy as writing a module with a couple of functions.

py.test runs on all POSIX operating systems and WINDOWS (XP/7/8) with Python versions 2.6 and above.

Installation

Use the following code to load the pytest module in the current Python distribution as well as a py.test.exe utility. Tests can be run using both.

```
pip install pytest
```

Usage

You can simply use the assert statement for asserting test expectations. pytest's assert introspection will intelligently report intermediate values of the assert expression freeing you from the need to learn the many names of **JUnit legacy methods**.

```
# content of test_sample.py
def func(x):
   return x + 1

def test_answer():
   assert func(3) == 5
```

Use the following command line to run the above test. Once the test is run, the following result is displayed on console −

```
C:\Python27>scripts\py.test -v test_sample.py
============================= test session starts ====================
platform win32 -- Python 2.7.9, pytest-2.9.1, py-1.4.31, pluggy-0.3.1 -- C:\Pyth
on27\python.exe
cachedir: .cache
rootdir: C:\Python27, inifile:
collected 1 items
test_sample.py::test_answer FAILED
================================= FAILURES ===================
_____________________________ test_answer ____________________________
   def test_answer():
>  assert func(3) == 5
E     assert 4 == 5
E     + where 4 = func(3)
test_sample.py:7: AssertionError
========================== 1 failed in 0.05 seconds ==================
```

The test can also be run from the command line by including pytest module using –m switch.

```
python -m pytest test_sample.py
```

Grouping Multiple Tests in a Class

Once you start to have more than a few tests it often makes sense to group tests logically, in classes and modules. Let’s write a class containing two tests −

```
class TestClass:
   def test_one(self):
      x = "this"
      assert 'h' in x
   def test_two(self):
      x = "hello"
      assert hasattr(x, 'check')
```

The following test result will be displayed −

```
C:\Python27>scripts\py.test -v test_class.py
============================= test session starts ====================
platform win32 -- Python 2.7.9, pytest-2.9.1, py-1.4.31, pluggy-0.3.1 -- C:\Pyt
on27\python.exe
```

```
cachedir: .cache
rootdir: C:\Python27, inifile:
collected 2 items
test_class.py::TestClass::test_one PASSED
test_class.py::TestClass::test_two FAILED
================================= FAILURES =====================
_________________________ TestClass.test_two ___________________________
self = <test_class.TestClass instance at 0x01309DA0>

   def test_two(self):
      x = "hello"
>  assert hasattr(x, 'check')
E     assert hasattr('hello', 'check')

test_class.py:7: AssertionError
==================== 1 failed, 1 passed in 0.06 seconds ======================
```