


PyTest Overview

Dec. 22, 2018 

A fully featured Cloud for the distributed Edge



Criteria for Choosing Test Framework

- Maintainability
- Debugability
- Flexibility
- Scalability

Easy to Write and Maintain Tests

- No-boilerplate (no test class instantiation)
- Simple syntax (asserts)
- Parametrizing tests (dynamically rename test, e.g., `remote_cli`)
- Skip test dynamically
- Strong support for test state management via setup/teardown hooks (4 scopes)
- Can scale up (system test) or down (unit test)

Flexibility in Test Execution

- Automatic collection of tests with decorator or string in test name
- Can ignore/deselect tests
- Can be called from python code

Strong Debugging Support

- Stop after first (N) failure optionally w. or w/o teardown
- Informative traceback
- Run pdb on failed tests
- Customizable xml and plain text reports

Backup

- Pass different data types to function param. E.g., None, list, dict, list of dict, function, etc.
- Can make use of existing openstack testcases. E.g., horizon

Comparison Chart

Test Framework	Main Advantages	Main Disadvantages	Best Application
Robot	<ol style="list-style-type: none"> 1. Use keyword driven strategy, where keywords are written in programming language such as python, and testcases are using non-programming language, which allows non-programmer to easily add test cases using existing keywords. 2. Pretty html report and log, and very good debugging info in the log. 	<ol style="list-style-type: none"> 1. New syntax needs to be learned to write test cases. 2. Limited support for what can be done, etc, no nested loop in test case, limited support for parametrizing test 3. Cannot skip a test 4. No global setup/teardown to share between multiple suites 	<ol style="list-style-type: none"> 1. Keyword driven acceptance tests 2. Coding skills vary among the team, where keywords can be developed by programmers while test case can be contributed by anyone using GUI/tsv/txt
Pytest	<ol style="list-style-type: none"> 1. Very good tracebacks when test fails. 2. Flexible and powerful test fixtures (setups/teardown). Other two frameworks only support 1 test setup 1 suite/class level setup, while for pytest, you can use multiple test fixtures at the same time. This allows sharing fixtures across test classes/modules. 3. Easy to customize the framework, such as putting custom info to the report. 4. no-boilerplate compared to Unittest 5. Can convert openstack testcases such as tempest test easily 	<ol style="list-style-type: none"> 1. Cannot easily control the order of the testcases 	<ol style="list-style-type: none"> 1. Anything does not require strict order of test cases
Unittest	<ol style="list-style-type: none"> 1. Included in python standard library 2. Probably minimal learning curve for most people in community 	<ol style="list-style-type: none"> 1. Overhead/boilerplate. Test class has to be defined for every test case, syntax not as simple as pytest 2. Does not support parametrized test 3. No global setup/teardown to share between multiple test classes 	<ol style="list-style-type: none"> 1. unit test and smaller scaled functional test cases