UnitTest Framework - Doctest API

The doctest API revolves around the following two container classes used to store interactive examples from docstrings −

- **Example** − A single Python statement, paired with its expected output.
- **DocTest** − A collection of Examples, typically extracted from a single docstring or a text file.

The following additional processing classes are defined to find, parse, and run, and check doctest examples −

- **DocTestFinder** − Finds all docstrings in a given module, and uses a DocTestParser to create a DocTest from every docstring that contains interactive examples.
- **DocTestParser** − Creates a doctest object from a string (such as an object’s docstring).
- **DocTestRunner** − Executes the examples in a doctest, and uses an OutputChecker to verify their output.
- **OutputChecker** − Compares the actual output from a doctest example with the expected output, and decides whether they match.

DocTestFinder Class

It is a processing class used to extract the doctests that are relevant to a given object, from its docstring and the docstrings of its contained objects. Doctests can currently be extracted from the following object types — modules, functions, classes, methods, staticmethods, classmethods, and properties.

This class defines the find() method. It returns a list of the DocTests that are defined by the *object‘s* docstring, or by any of its contained objects’ docstrings.

DocTestParser Class

It is a processing class used to extract interactive examples from a string, and use them to create a DocTest object. This class defines the following methods −

- **get_doctest()** − Extract all doctest examples from the given string, and collect them into a **DocTest** object.

- **get_examples(string[, name])** − Extract all doctest examples from the given string, and return them as a list of **Example** objects. Line numbers are 0-based. The optional argument name is a name identifying this string, and is only used for error messages.
- **parse(string[, name])** − Divide the given string into examples and intervening text, and return them as a list of alternating **Examples** and strings. Line numbers for the **Examples** are 0-based. The optional argument name is a name identifying this string, and is only used for error messages.

DocTestRunner Class

This is a processing class used to execute and verify the interactive examples in a DocTest. The following methods are defined in it −

report_start()

Report that the test runner is about to process the given example. This method is provided to allow subclasses of **DocTestRunner** to customize their output; it should not be called directly

report_success()

Report that the given example ran successfully. This method is provided to allow subclasses of DocTestRunner to customize their output; it should not be called directly.

report_failure()

Report that the given example failed. This method is provided to allow subclasses of **DocTestRunner** to customize their output; it should not be called directly.

report_unexpected_exception()

Report that the given example raised an unexpected exception. This method is provided to allow subclasses of DocTestRunner to customize their output; it should not be called directly.

run(test)

Run the examples in *test* (a DocTest object), and display the results using the writer function *out*.

summarize([verbose])

Print a summary of all the test cases that have been run by this DocTestRunner, and return a *named tuple* TestResults(failed, attempted). The optional *verbose* argument controls how detailed the summary is. If the verbosity is not specified, then the DocTestRunner‘s verbosity is used.

OutputChecker Class

This class is used to check whether the actual output from a doctest example matches the expected output.

The following methods are defined in this class −

check_output()

Return **True** if the actual output from an example (*got*) matches with the expected output (*want*). These strings are always considered to match if they are identical; but depending on what option flags the test runner is using, several non-exact match types are also possible. See section *Option Flags* and *Directives* for more information about option flags.

output_difference()

Return a string describing the differences between the expected output for a given example (*example*) and the actual output (*got*).

DocTest Integration with Unittest

The doctest module provides two functions that can be used to create unittest test suites from modules and text files containing doctests. To integrate with unittest test discovery, include a load_tests() function in your test module −

```
import unittest
import doctest
import doctestexample

def load_tests(loader, tests, ignore):
   tests.addTests(doctest.DocTestSuite(doctestexample))
   return tests
```

A combined TestSuite of tests from unittest as well as doctest will be formed and it can now be executed by unittest module's main() method or run() method.

The following are the two main functions for creating **unittest.TestSuite** instances from text files and modules with the doctests −

doctest.DocFileSuite()

It is used to convert doctest tests from one or more text files to a **unittest.TestSuite**. The returned unittest.TestSuite is to be run by the unittest framework and runs the interactive examples in each file. If any of the examples in a file fails, then the synthesized unit test fails, and a **failureExcept** exception is raised showing the name of the file containing the test and a (sometimes approximate) line number.

doctest.DocTestSuite()

It is used to convert doctest tests for a module to a **unittest.TestSuite**.

The returned unittest.TestSuite is to be run by the unittest framework and runs each doctest in the module. If any of the doctests fail, then the synthesized unit test fails, and a **failureException** exception is raised showing the name of the file containing the test and a (sometimes approximate) line number

Under the covers, DocTestSuite() creates a **unittest.TestSuite** out of doctest.DocTestCase instances, and DocTestCase is a subclass of unittest.TestCase.

Similarly, DocFileSuite() creates a unittest.TestSuite out of doctest.DocFileCase instances, and DocFileCase is a subclass of DocTestCase.

So both ways of creating a unittest.TestSuite run instances of DocTestCase. When you run doctest functions yourself, you can control the doctest options in use directly, by passing option flags to doctest functions.

However, if you’re writing a unittest framework, unittest ultimately controls when and how the tests get run. The framework author typically wants to control doctest reporting options (perhaps, e.g., specified by command line options), but there’s no way to pass options through unittest to doctest test runners.