UnitTest Framework - Doctest

Python' standard distribution contains 'Doctest' module. This module's functionality makes it possible to search for pieces of text that look like interactive Python sessions, and executes these sessions to see if they work exactly as shown.

Doctest can be very useful in the following scenarios −

- To check that a module’s docstrings are up-to-date by verifying that all interactive examples still work as documented.
- To perform regression testing by verifying that interactive examples from a test file or a test object work as expected.
- To write tutorial documentation for a package, liberally illustrated with input-output examples

In Python, a 'docstring' is a string literal which appears as the first expression in a class, function or module. It is ignored when the suite is executed, but it is recognized by the compiler and put into the **__doc__** attribute of the enclosing class, function or module. Since it is available via introspection, it is the canonical place for documentation of the object.

It is a usual practice to put example usage of different parts of Python code inside the docstring. The doctest module allows to verify that these docstrings are up-to-date with the intermittent revisions in code.

In the following code, a factorial function is defined interspersed with example usage. In order to verify if the example usage is correct, call the testmod() function in doctest module.

```
"""
This is the "example" module.

The example module supplies one function, factorial(). For example,

>>> factorial(5)
120
"""

def factorial(x):
   """Return the factorial of n, an exact integer >= 0.
   >>> factorial(-1)
   Traceback (most recent call last):
       ...
   ValueError: x must be >= 0
   """

   if not x >= 0:
      raise ValueError("x must be >= 0")
   f = 1
   for i in range(1,x+1):
      f = f*i
   return f

if __name__ == "__main__":
   import doctest
   doctest.testmod()
```

Enter and save the above script as FactDocTest.py and try to execute this script from the command line.

```
Python FactDocTest.py
```

No output will be shown unless the example fails. Now, change the command line to the following −

```
Python FactDocTest.py -v
```

The console will now show the following output −

```
C:\Python27>python FactDocTest.py -v
Trying:
   factorial(5)
Expecting:
   120
ok
Trying:
   factorial(-1)
Expecting:
   Traceback (most recent call last):
      ...
   ValueError: x must be >= 0
ok
2 items passed all tests:
   1 tests in __main__
   1 tests in __main__.factorial
```

```
2 tests in 2 items.
2 passed and 0 failed.
Test passed.
```

If, on the other hand, the code of factorial() function doesn't give expected result in docstring, failure result will be displayed. For instance, change f = 2 in place of f = 1 in the above script and run the doctest again. The result will be as follows −

```
Trying:
   factorial(5)
Expecting:
   120
**********************************************************************
File "docfacttest.py", line 6, in __main__
Failed example:
factorial(5)
Expected:
   120
Got:
   240
Trying:
   factorial(-1)
Expecting:
   Traceback (most recent call last):
      ...
   ValueError: x must be >= 0
ok
1 items passed all tests:
   1 tests in __main__.factorial
**********************************************************************
1 items had failures:
   1 of 1 in __main__
2 tests in 2 items.
1 passed and 1 failed.
***Test Failed*** 1 failures.
```

Doctest: Checking Examples in a Text File

Another simple application of doctest is testing interactive examples in a text file. This can be done with the testfile() function.

The following text is stored in a text file named 'example.txt'.

```
Using ''factorial''
-------------------
This is an example text file in reStructuredText format. First import
''factorial'' from the ''example'' module:
   >>> from example import factorial
Now use it:
   >>> factorial(5)
   120
```

The file content is treated as docstring. In order to verify the examples in the text file, use the testfile() function of doctest module.

```
def factorial(x):
   if not x >= 0:
      raise ValueError("x must be >= 0")
   f = 1
   for i in range(1,x+1):
      f = f*i
   return f

if __name__ == "__main__":
   import doctest
   doctest.testfile("example.txt")
```

- As with the testmod(), testfile() won't display anything unless an example fails. If an example does fail, then the failing example(s) and the cause(s) of the failure(s) are printed to console, using the same format as testmod().
- In most cases a copy-and-paste of an interactive console session works fine, but doctest isn't trying to do an exact emulation of any specific Python shell.
- Any expected output must immediately follow the final '>>> ' or '... ' line containing the code, and the expected output (if any) extends to the next '>>> ' or all-whitespace line.
- Expected output cannot contain an all-whitespace line, since such a line is taken to signal the end of expected output. If expected output does contain a blank line, put <BLANKLINE> in your doctest example each place a blank line is expected.