Amazing Python Guide - The Hitchhiker's Guide to Python!!

Online Guide

- GitHub guide
- Structure your project
- Code Style
- Reading Great Code
- Gotchas
- Logging

Testing Your Code

Testing

- Use long and descriptive names for testing functions. The style guide here is slightly different than that of running code, where short names are often preferred. The reason is testing functions are never called explicitly. square() or even sqr() is ok in running code, but in testing code you would have names such as test_square_of_number_2() , test_square_negative_number() . These function names are displayed when a test fails, and should be as descriptive as possible.
- Each test unit must be fully independent. Each test must be able to run alone, and also within the test suite, regardless of the order that they are called. The implication of this rule is that each test must be loaded with a fresh dataset and may have to do some cleanup afterwards. This is usually handled by :meth:`setUp()` and :meth:`tearDown()` methods

Python unittest Test Discovery

https://docs.python.org/3/library/unittest.html#unittest-test-discovery

How UnitTest Works: main = TestProgram

https://stackoverflow.com/questions/20992731/meaning-of-unittest-main-in-python-unittest-module

- when you execute unittest.main(), an object of TestProgram gets created which calls self.runTests() at line 768.

- The constructor also takes your current file as the default module containing the tests (module='**main**').
- When runTests() is called, it in turn calls self.testrunner.run().
- When you refer to the "run" method of TextTestRunner class, you will find that it actually runs and reports all your test results.
- Test discovery is done by TestProgram.parseArgs at line 775 when you call unittest.main().
- self.createTests at line 798 is actually responsible for discovering all your test cases and creating a test suite. This is all the magic.
- Hangling further down you will find the loop which is used to find all test cases in loader.py in the methods loadTestsFromModule() and loadTestsFromName().
- It iterates over all elements in a module and applies a check issubclass(obj, case.TestCase) to find the test cases.

Python Guide - Testing your code

http://python-guide-pt-br.readthedocs.io/en/latest/writing/tests/

Parameterized Python test framework

https://github.com/wolever/parameterized

Running unittest in Jupyter

SO Run UnitTest in Jupyter

- unittest.main looks at sys.argv by default, which is what started IPython, hence the error about the kernel connection file not being a valid attribute. You can pass an explicit list to main to avoid looking up sys.argv.
- In the notebook, you will also want to include **exit=False** to prevent unittest.main from trying to shutdown the kernel process:

```
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

- You can pass further arguments in the argv list, e.g.

```
unittest.main(argv=['ignored', '-v'], exit=False)
```

In [1]:

```
import unittest

def fun(x):
    return x + 1

class MyTest(unittest.TestCase):
    def test(self):
        self.assertEqual(fun(3), 4)
```

14 different slides on COMPLEX UNIT TESTING

http://starship.python.net/crew/tbryan/UnitTestTalk/slide12.html

from mx.DateTime import DateTime

import unittest

import DateRange

class TestDateRange(unittest.TestCase):

```
    def test_creation_basic( self ):
        A = DateTime( 2002, 1, 1 )
        B = DateTime( 2002, 2, 1 )
        range = DateRange.DateRange( A, B )
        self.assertEquals( range.start, A )
        self.assertEquals( range.end, B )
```

if **name** == '**main**': unittest.main()