tdd_advanced

September 29, 2016

1 Advanced topics in test driven development

SDES course
Prabhu Ramachandran
September 2016

1.1 Introduction

- Already seen the basics
- Learn some advanced topics
- `hypothesis`
- `unittest`
- `coverage`
- `mock`
- `pytest`
- Odds and ends

1.2 The hypothesis package

- http://hypothesis.readthedocs.io
- `pip install hypothesis`
- General idea for testing:
- Make test data
- Perform operations
- Assert something after operation

1.2.1 Overview

- Hypothesis automates this!
- Describe range of scenarios
- Computer explores these and tests them
- With hypothesis:

- Generate random data using specification
- Perform operations
- assert something about result

1.2.2 Example

```
In [ ]: from hypothesis import given
        from hypothesis import strategies as st

        def gcd(a, b):
            if b == 0:
                return a
            return gcd(b, a % b)

        @given(st.integers(min_value=0), st.integers(min_value=0))
        def test_gcd(a, b):
            result = gcd(a, b)
            # Now what?
            # assert a%result == 0
```

1.2.3 Example: adding a specific case

This is meaningless here but illustrative:

```
In [ ]: @given(st.integers(min_value=0), st.integers(min_value=0))
        @example(a=44, b=19)
        def test_gcd(a, b):
            result = gcd(a, b)
            # Now what?
            # assert a%result == 0
```

1.2.4 More details

- `given` generates inputs
- `strategies`: provides a strategy for inputs
- Different stratiegies
  - `integers`
  - `floats`
  - `text`
  - `booleans`
  - `tuples`
  - `lists`
  - ...
- See: http://hypothesis.readthedocs.io/en/latest/data.html

1.2.5 Example exercise

- Write a simple run-length encoding function called `encode`
- Write another called `decode` to produce the same input from the output of `encode`

For example:

```
In [ ]: def encode(text):
            return []
        def decode(lst):
            return ''
```

1.2.6 The test

```
In [ ]: from hypothesis import given
        from hypothesis import strategies as st

        @given(st.text())
        def test_decode_inverts_encode(s):
            assert decode(encode(s)) == s
```

1.2.7 Summary

- Different approach to testing
- hypothesis does the hard work
- Can do a lot more!
- Read the docs for more
- For some detailed articles: http://hypothesis.works/articles/intro/
- Here in particular is one interesting article: http://hypothesis.works/articles/calculating-the-mean/

1.3 Unittest module

- Basic idea and style is from JUnit
- Some consider this old style
- Important to understand how it works

1.3.1 How to use unittest

- Built into standard library
- Subclass `unittest.TestCase`
- Create test methods

1.3.2 A simple example

Let us test `gcd.py` with unittest

```
In [ ]: # gcd.py
        def gcd(a, b):
            if b == 0:
```

```
            return a
        return gcd(b, a%b)
```

1.3.3 Writing the test

```
In [ ]: # test_gcd.py
        import unittest

        #from gcd import gcd

        class TestGCD(unittest.TestCase):
            def test_gcd_works_for_positive_integers(self):
                self.assertEqual(gcd(48, 64), 16)
                self.assertEqual(gcd(44, 19), 1)

        if __name__ == '__main__':
            unittest.main()
```

1.3.4 Running it

- Just run `python test_gcd.py`
- Also works with `nosetests` and `pytest`

1.3.5 Notes

- Note the name of the method
- Note the use of `self.assertEqual`
- Also available: `assertNotEqual`, `assertTrue`, `assertFalse`, `assertIs`, `assertIsNot`
- `assertIsNone`, `assertIn`, `assertIsInstance`, `assertRaises`
- `assertAlmostEqual`, `assertListEqual`, `assertSequenceEqual` ...
- https://docs.python.org/2/library/unittest.html

1.3.6 Fixtures

- What if you want to do something common before all tests?
- Typically called a **fixture**
- Use the `setUp` and `tearDown` methods for method-level fixtures

1.3.7 Silly fixture example

```
In [ ]: # test_gcd.py
        import gcd
        import unittest
```

```
class TestGCD(unittest.TestCase):
    def setUp(self):
        print("setUp")
    def tearDown(self):
        print("tearDown")
    def test_gcd_works_for_positive_integers(self):
        self.assertEqual(gcd(48, 64), 16)
        self.assertEqual(gcd(44, 19), 1)

if __name__ == '__main__':
    unittest.main()
```

1.3.8 Exercise

- Fix bug with negative numbers in gcd.py.
- Use TDD.

1.3.9 Using hypothesis with unittest

```
In [ ]: # test_rle.py
        from hypothesis import given, example
        from hypothesis import strategies as st

        #from rle import decode, encode
        import unittest

        class TestRLE(unittest.TestCase):
            @given(st.text())
            @example('')
            def test_decode_inverts_encode(self, s):
                assert decode(encode(s)) == s

        if __name__ == '__main__':
            unittest.main()
```

1.3.10 Some notes on style

- Use descriptive function names
- Intent matters
- Segregate the test code into the following
  - Given: what is the context of the test?
  - When: what action is taken to actually test the problem
  - Then: what do we actually ensure.

1.3.11 More on intent driven programming

> Programs must be written for people to read, and only incidentally for machines to execute. – Harold Abelson

The code should make the intent clear. For example:

```
In [ ]: if self.temperature > 600 and self.pressure > 10e5:
            message = 'hello you have a problem here!'
            message += 'current temp is %s'%(self.temperature)
            print(message)
            self.valve.close()
            self.raise_warning()
            self.shutdown()
```

is totally unclear as to the intent. Instead refactor as follows:

```
In [ ]: if self.reactor_is_critical():
            self.shutdown_with_warning()
```

1.3.12 A more involved testing example

- Motivational problem:

  Find all the git repositories inside a given directory recursively. Make this a command line tool supporting command line use.

- Write tests for the code
- Some rules:

0. The test should be runnable by anyone (even by a computer), almost anywhere.
1. Don't write anything in the current directory (use a temporary directory).
2. Cleanup any files you create while testing.
3. Make sure tests do not affect global state too much.

1.3.13 Solution

1. Create some test data.
2. Test!
3. Cleanup the test data

1.3.14 Class-level fixtures

- Use `setupClass` and `tearDownClass` classmethods for class level fixtures.

1.3.15 Module-level fixtures

- `setup_module`, `teardown_module`
- This is nose specific
- Can be used for a module-level fixture
- http://nose.readthedocs.io/en/latest/writing_tests.html

1.4 Coverage

- Assess the amount of code that is covered by testing
- http://coverage.readthedocs.io/
- `pip install coverage`
- Integrates with nosetests/pytest

1.4.1 Typical coverage usage

```
coverage run -m nose.core my_package
coverage report -m
coverage html
```

1.5 mock package

- Motivational examples
- Example: reading some twitter data
- Example: function to post an update to facebook or twitter
- Example: email user when simulation crashes
- Can you test it? How?

1.5.1 Using mock: the big picture

- Do you really want to post something on facebook?
- Or do you want to know if the right method was called with the right arguments?
- Idea: "mock" the objects that do something and test them
- Quoting from the docs:

  It allows you to replace parts of your system under test with mock objects and make assertions about how they have been used.

1.5.2 Installation

- Built-in on Python >= 3.3
  - `from unittest import mock`
- else `pip install mock`
  - `import mock`

1.5.3 Simple examples

Say we have a class:

```
In [ ]: class ProductionClass(object):
            def method(self, *args):
                # We *do not* want to run this in the test
                pass
```

To mock the `ProductionClass.method` do this:

```
In [ ]: from unittest.mock import MagicMock
        thing = ProductionClass()
        thing.method = MagicMock(return_value=3)
        thing.method(3, 4, 5, key='value')
        thing.method.assert_called_with(3, 4, 5, key='value')
```

1.5.4 More practical use case

- Mocking a module or system call
- Mocking an object or method
- Remember that after testing you want to restore original state
- Use `mock.patch`

1.5.5 An example

- Write code to remove generated files from LaTeX compilation, i.e. remove the `*.aux`, `*.log`, `*.pdf` etc.

Here is a simple attempt:

```
In [ ]: # clean_tex.py
        import os

        def cleanup(tex_file_pth):
            base = os.path.splitext(tex_file_pth)[0]
            for ext in ('.aux', '.log'):
                f = base + ext
                if os.path.exists(f):
                    os.remove(f)
```

1.5.6 Testing this with mock

```
In [ ]: import mock

        @mock.patch('clean_tex.os.remove')
        def test_cleanup_removes_extra_files(mock_remove):
            cleanup('foo.tex')

            expected = [mock.call('foo.' + x) for x in ('aux', 'log')]

            mock_remove.assert_has_calls(expected)
```

1.5.7 Notes

- Note the mocked argument that is passed.
- Note that we did not mock `os.remove`
- Mock where the object is looked up

1.5.8 Doing more

```
In [ ]: import mock

        @mock.patch('clean_tex.os.path')
        @mock.patch('clean_tex.os.remove')
        def test_cleanup_does_not_fail_when_files_dont_exist(mock_remove, mock_path
            # Setup the mock_path to return False
            mock_path.exists.return_value = False

            cleanup('foo.tex')

            mock_remove.assert_not_called()
```

- Note the order of the passed arguments
- Note the name of the method

1.5.9 Patching instance methods

Use `mock.patch.object` to patch an instance method

```
In [ ]: @mock.patch.object(ProductionClass, 'method')
        def test_method(mock_method):
            obj = ProductionClass()
            obj.method(1)

            mock_method.assert_called_once_with(1)
```

Mock works as a context manager:

```
In [ ]: with mock.patch.object(ProductionClass, 'method') as mock_method:
            obj = ProductionClass()
            obj.method(1)

        mock_method.assert_called_once_with(1)
```

1.5.10 More articles on mock

- See more here https://docs.python.org/3/library/unittest.mock.html
- https://www.toptal.com/python/an-introduction-to-mocking-in-python

1.6 Pytest

- http://doc.pytest.org/
- Offers many useful and convenient features
- Detailed info from simple assert statements
- Automatic discovery of tests
- Modular fixures
- To find all fixtures `pytest --fixtures`

1.6.1 Basic features of pytest

```
In [ ]: #from gcd import gcd
        import pytest

        def test_gcd():
            assert gcd(44, 19) == 1

        import pytest
        def test_casting_raises_value_error():
            with pytest.raises(ValueError):
                int('asd')

        def test_floating_point():
            assert 0.1 + 0.2 == pytest.approx(0.3)
```

1.6.2 Grouping tests

```
In [ ]: class TestClass:

            def test_something(self):
                assert 1 == 1

            def test_something_else(self):
                assert 2 == 2
```

1.6.3 Builtin fixtures: tmpdir

```
In [ ]: import os
        def test_create_file(tmpdir):
            p = tmpdir.mkdir("sub").join("hello.txt")
            p.write("content")
            assert p.read() == "content"
            assert len(tmpdir.listdir()) == 1
            assert 0
```

1.6.4 Builtin fixtures: monkeypatch

Like mock but slightly different.

```
In [2]: import os.path
        def getssh(): # pseudo application code
            return os.path.join(os.path.expanduser("~admin"), '.ssh')

        def test_mytest(monkeypatch):
            def mockreturn(path):
                return '/abc'
            monkeypatch.setattr(os.path, 'expanduser', mockreturn)
            x = getssh()
            assert x == '/abc/.ssh'
```

1.6.5 Builtin fixtures: capture output

```
In [ ]: def test_myoutput(capsys):
            # or use "capfd" for fd-level
            print("hello")
            sys.stderr.write("world\n")
            out, err = capsys.readouterr()
            assert out == "hello\n"
            assert err == "world\n"
            print("next")
            out, err = capsys.readouterr()
            assert out == "next\n"
```

1.6.6 Easy to create your own fixtures

```
In [ ]: import pytest

        @pytest.fixture
        def smtp():
            import smtplib
            return smtplib.SMTP("smtp.gmail.com")

        def test_ehlo(smtp):
            response, msg = smtp.ehlo()
            assert response == 250
            assert 0 # for demo purposes
```

- Use `@pytest.fixture(scope='module')`
- scope can be module, function, session
- Can `yield` in the fixture for cleanup

1.7 Odds and ends

1.7.1 Linters

- `pyflakes`
- `flake8`

- Very important to use.
- Integrates with good editors

1.7.2 IPython goodies

- `%run`
- Debug with `%run`
- Profiling
- `%pdb`
- `%debug`
- `pdb.set_trace()`
- IPython set trace: `from IPython.core.debugger import Tracer; Tracer()()`
- See here: http://www.scipy-lectures.org/advanced/debugging/

Thats all folks!
Thank you!