

Coding by Behavior Driven Design

Examples of workin' with Gherkin



A.S. Rustenburg, Memorial Sloan Kettering Cancer Center
Chodera lab group talk, October 17 2014

Table of Contents

- 1 Why test code?
- 2 How do we test code in python?
Unit testing
- 3 Good testing practices
- 4 Alternate Testing Paradigms
Test-Driven Development
Behavior-Driven Development
Cucumber and Behave

Table of Contents

1 Why test code?

2 How do we test code in python?

Unit testing

3 Good testing practices

4 Alternate Testing Paradigms

Test-Driven Development

Behavior-Driven Development

Cucumber and Behave

Why do we write tests for code?

Writing tests for code can prevent mistakes, and guide development of software.

Common reasons for writing tests

- ▶ Catch bugs introduced during development
- ▶ Maintain backwards compatibility
- ▶ Verify output of newly implemented function
- ▶ Introduction to new developers
- ▶ Provide a scaffold for code design

Why do we write tests for code?

Writing tests for code can prevent mistakes, and guide development of software.

Common reasons for writing tests

- ▶ Catch bugs introduced during development
- ▶ Maintain backwards compatibility
- ▶ Verify output of newly implemented function
- ▶ Introduction to new developers
- ▶ Provide a scaffold for code design

Why do we write tests for code?

Writing tests for code can prevent mistakes, and guide development of software.

Common reasons for writing tests

- ▶ Catch bugs introduced during development
- ▶ Maintain backwards compatibility
- ▶ Verify output of newly implemented function
- ▶ Introduction to new developers
- ▶ Provide a scaffold for code design

Why do we write tests for code?

Writing tests for code can prevent mistakes, and guide development of software.

Common reasons for writing tests

- ▶ Catch bugs introduced during development
- ▶ Maintain backwards compatibility
- ▶ Verify output of newly implemented function
- ▶ Introduction to new developers
- ▶ Provide a scaffold for code design

Why do we write tests for code?

Writing tests for code can prevent mistakes, and guide development of software.

Common reasons for writing tests

- ▶ Catch bugs introduced during development
- ▶ Maintain backwards compatibility
- ▶ Verify output of newly implemented function
- ▶ Introduction to new developers
- ▶ Provide a scaffold for code design

Why do we write tests for code?

Writing tests for code can prevent mistakes, and guide development of software.

Common reasons for writing tests

- ▶ Catch bugs introduced during development
- ▶ Maintain backwards compatibility
- ▶ Verify output of newly implemented function
- ▶ Introduction to new developers
- ▶ Provide a scaffold for code design

Table of Contents

- 1 Why test code?
- 2 How do we test code in python?
Unit testing
- 3 Good testing practices
- 4 Alternate Testing Paradigms
Test-Driven Development
Behavior-Driven Development
Cucumber and Behave

How to test code?

Example python libraries

unittest

A solid testing library that provides many functions to help you test units of code.

```
# unittest_example.py
import unittest

def my_function(x, y):
    """Subtract y from x"""
    return x - y

class MyTest(unittest.TestCase):
    def test_my_function_1(self):
        self.assertEqual(my_function(7, 4), 3)
    def test_my_function_2(self):
        self.assertRaises(TypeError, my_function, [7, '4'])
if __name__ == '__main__':
    unittest.main()
```

<https://docs.python.org/2/library/unittest.html>

How to test code?

Example python libraries

doctest

Write test in your documentation strings.

```
# doctest_example.py
def my_function(x, y):
    """Subtract y from x
    Examples
    -----
    >>> my_function(7, 4)
    3
    >>> my_function(7, '4')
    Traceback (most recent call last):
    ...
    TypeError: unsupported operand type(s) for -: 'int' and 'str'
    """
    return x - y

if __name__ == "__main__":
    #Execute script as: python doctest_example.py -v
    import doctest
    doctest.testmod()
```

<https://docs.python.org/2/library/doctest.html>

How to test code?

Example python libraries

nosetests

Adds test discovery on top of available tools.

```
# test_nose_example.py
def my_function(x, y):
    """Subtract y from x"""
    return x - y

def test_my_function():
    assert my_function(7,4) == 3
```

Try commands such as:

- ▶ nosetests
- ▶ nosetests unittest_example.py
- ▶ nosetests --with-doctest

Table of Contents

- 1 Why test code?
- 2 How do we test code in python?
Unit testing
- 3 Good testing practices
- 4 Alternate Testing Paradigms
Test-Driven Development
Behavior-Driven Development
Cucumber and Behave

Good testing practices

Some things to keep in mind when writing tests.

Attempt to at least:

- ▶ Focus on small units of functionality and ensure correctness
- ▶ Tests must be able to run independently (see Mock for help)
- ▶ Keep them fast! Slowness discourages running tests.
- ▶ Write a test for debugging.
- ▶ Keep your code clear, keep your tests clearer!

Inspired by: The Hitchhiker's Guide to Python

<https://github.com/kennethreitz/python-guide>

Good testing practices

Some things to keep in mind when writing tests.

Attempt to at least:

- ▶ Focus on small units of functionality and ensure correctness
- ▶ Tests must be able to run independently (see Mock for help)
- ▶ Keep them fast! Slowness discourages running tests.
- ▶ Write a test for debugging.
- ▶ Keep your code clear, keep your tests clearer!

Inspired by: The Hitchhiker's Guide to Python

<https://github.com/kennethreitz/python-guide>

Good testing practices

Some things to keep in mind when writing tests.

Attempt to at least:

- ▶ Focus on small units of functionality and ensure correctness
- ▶ Tests must be able to run independently (see Mock for help)
- ▶ Keep them fast! Slowness discourages running tests.
- ▶ Write a test for debugging.
- ▶ Keep your code clear, keep your tests clearer!

Inspired by: The Hitchhiker's Guide to Python

<https://github.com/kennethreitz/python-guide>

Good testing practices

Some things to keep in mind when writing tests.

Attempt to at least:

- ▶ Focus on small units of functionality and ensure correctness
- ▶ Tests must be able to run independently (see Mock for help)
- ▶ Keep them fast! Slowness discourages running tests.
- ▶ Write a test for debugging.
- ▶ Keep your code clear, keep your tests clearer!

Inspired by: The Hitchhiker's Guide to Python

<https://github.com/kennethreitz/python-guide>

Good testing practices

Some things to keep in mind when writing tests.

Attempt to at least:

- ▶ Focus on small units of functionality and ensure correctness
- ▶ Tests must be able to run independently (see Mock for help)
- ▶ Keep them fast! Slowness discourages running tests.
- ▶ Write a test for debugging.
- ▶ Keep your code clear, keep your tests clearer!

Inspired by: The Hitchhiker's Guide to Python

<https://github.com/kennethreitz/python-guide>

Good testing practices

Some things to keep in mind when writing tests.

Attempt to at least:

- ▶ Focus on small units of functionality and ensure correctness
- ▶ Tests must be able to run independently (see Mock for help)
- ▶ Keep them fast! Slowness discourages running tests.
- ▶ Write a test for debugging.
- ▶ **Keep your code clear, keep your tests clearer!**

Inspired by: The Hitchhiker's Guide to Python

<https://github.com/kennethreitz/python-guide>

Good testing practices

Some things to keep in mind when writing tests.

Attempt to at least:

- ▶ Focus on small units of functionality and ensure correctness
- ▶ Tests must be able to run independently (see Mock for help)
- ▶ Keep them fast! Slowness discourages running tests.
- ▶ Write a test for debugging.
- ▶ Keep your code clear, keep your tests clearer!

Inspired by: The Hitchhiker's Guide to Python

<https://github.com/kennethreitz/python-guide>

Mock, a useful tool in testing

Keeping your tests independent.

Mock

Making sure that a method was called:

```
# mock_example.py
from mock import Mock

class Myclass(object):
    def closer(self, something):
        something.close()

mycase = Myclass()
mock = Mock()
mycase.closer(mock)
mock.close.assert_called_with()
```

<http://www.voidspace.org.uk/python/mock/index.html>

Table of Contents

- 1 Why test code?
- 2 How do we test code in python?
Unit testing
- 3 Good testing practices
- 4 Alternate Testing Paradigms
Test-Driven Development
Behavior-Driven Development
Cucumber and Behave

Test-Driven Development

Developing code starting with tests

How does it differ from conventional unit testing?

- ▶ Instead of units of code, one first writes tests of the unit
- ▶ Then, you fill in the code to make the tests pass
- ▶ Finally, verify that the test succeeds.

Test-Driven Development

Developing code starting with tests

What are the advantages?

- ▶ Keeps code concise
- ▶ Clear purpose for implemented code
- ▶ **Keep It Simple Stupid!**
- ▶ **You Aren't Gonna Need It.**

Table of Contents

- 1 Why test code?
- 2 How do we test code in python?
Unit testing
- 3 Good testing practices
- 4 Alternate Testing Paradigms
Test-Driven Development
Behavior-Driven Development
Cucumber and Behave

Behavior-Driven Development

Developing from the end-users perspective.

What are the key ideas?

- ▶ The intended behavior of a program determines the development strategy
- ▶ From the start, you decide *what* should be tested.
- ▶ Using object-oriented natural language to describe behavior
- ▶ Allow non-experts to design software and tests
- ▶ Then have programmers implement them

Behavior-Driven Development

Developing from the end-users perspective.

What are the key ideas?

- ▶ The intended behavior of a program determines the development strategy
- ▶ From the start, you decide *what* should be tested.
- ▶ Using object-oriented natural language to describe behavior
- ▶ Allow non-experts to design software and tests
- ▶ Then have programmers implement them

Behavior-Driven Development

Developing from the end-users perspective.

What are the key ideas?

- ▶ The intended behavior of a program determines the development strategy
- ▶ From the start, you decide *what* should be tested.
- ▶ Using object-oriented natural language to describe behavior
- ▶ Allow non-experts to design software and tests
- ▶ Then have programmers implement them

Behavior-Driven Development

Developing from the end-users perspective.

What are the key ideas?

- ▶ The intended behavior of a program determines the development strategy
- ▶ From the start, you decide *what* should be tested.
- ▶ Using object-oriented natural language to describe behavior
- ▶ Allow non-experts to design software and tests
- ▶ Then have programmers implement them

Behavior-Driven Development

Developing from the end-users perspective.

What are the key ideas?

- ▶ The intended behavior of a program determines the development strategy
- ▶ From the start, you decide *what* should be tested.
- ▶ Using object-oriented natural language to describe behavior
- ▶ Allow non-experts to design software and tests
- ▶ Then have programmers implement them

Behavior-Driven Development

Developing from the end-users perspective.

What are the key ideas?

- ▶ The intended behavior of a program determines the development strategy
- ▶ From the start, you decide *what* should be tested.
- ▶ Using object-oriented natural language to describe behavior
- ▶ Allow non-experts to design software and tests
- ▶ Then have programmers implement them

What are they all about?

- ▶ Cucumber (Ruby), Behave(Python), Specflow (.NET) et al.
- ▶ The behavior of the program is written down in a business readable, domain specific language called *Gherkin*
- ▶ This is then parsed and matched to functions called *step functions*
- ▶ The step functions are written in actual programming languages
- ▶ They test other underlying code and take input from the Gherkin parser

<https://github.com/cucumber/cucumber/wiki/Gherkin>

A simple example

```
# example.feature
```

```
Feature: An example of behavior driven design
```

```
Scenario: I want to subtract two numbers
```

```
Given our subtractor is installed
```

```
When I subtract 4 from 7
```

```
Then the result should be 3
```

Step methods

The underlying python code

A simple example

```
# Example steps
from behave import given, when, then

# Given our subtractor is installed
@given('our subtractor is installed')
def step_impl(context):
    pass

# When I subtract 4 from 7
@when('I subtract {y} from {x}')
def step_impl(context, y, x):
    context.result = int(x) - int(y)

# Then the result should be 3
@then('the result should be {result}')
def step_impl(context, result):
    assert context.result == int(result)
```

Useful sources of information

..the wiki is not enough..

Urls

- ▶ Hitchhiker's Guide to Python :
<https://github.com/kennethreitz/python-guide>
- ▶ doctest, unittest: <https://docs.python.org/2/library/development.html>
- ▶ nose: <https://nose.readthedocs.org>
- ▶ mock: <http://www.voidspace.org.uk/python/mock/mock.html>
- ▶ behave: <http://pythonhosted.org/behave/>
- ▶ cucumber: <http://cukes.info>