Python wrap cases Documentation

Release 0.1.8

Kirill Ermolov

August 21, 2015

Contents

Contents:

CHAPTER 1

Python wrap cases

Simple library for generate test cases with parameters.

1.1 What is this?

This library helps to generate tests with parameters.

Let's write some tests for this function:

```
import re


def clear_start_end_dash(string):
    return re.sub(r'^[\s\-]*-|-[\s\-]*$', '', string)
```

We may write something like this:

```
from unittest import TestCase


class ClearStartEndDashTest(TestCase):

    def test_remove_first_dash(self):
        result = clear_start_end_dash('-my string')
        self.assertEqual(result, 'my string')

    def test_remove_all_first_dashes(self):
        result = clear_start_end_dash('-  -- --my string')
        self.assertEqual(result, 'my string')

    def test_remove_last_dash(self):
        result = clear_start_end_dash('my string-')
        self.assertEqual(result, 'my string')

    def test_remove_all_last_dashes(self):
        result = clear_start_end_dash('my string-- -- -- - ')
        self.assertEqual(result, 'my string')

    def test_keep_dash_at_center(self):
        result = clear_start_end_dash('my-string')
        self.assertEqual(result, 'my-string')
```

It's good, but we spent a lot of time to write those absolutely the same test functions.

So let's decrease the number of duplicate functions:

```
from unittest import TestCase


class ClearStartEndDashDryTest(TestCase):

    def test_remove_dash(self):
        cases = (
            ('-my string', 'my string'),
            ('-  -- --my string', 'my string'),
            ('my string-', 'my string'),
            ('my string-- -- -- - ', 'my string'),
            ('my-string', 'my-string')
        )
        for string, expected_result in cases:
            result = clear_start_end_dash(string)
            self.assertEqual(result, expected_result)
```

This code has a few problems:

- Easy to write but difficult to read.
- We can't use test fixture (*setUp*, *tearDown*) with each case.
- If some case fails, the other cases won't run.
- If test *test_remove_dash* fails, it won't help us find out what happened.

Look how easy we may solve these problems using this library:

```
from unittest import TestCase
from python_wrap_cases import wrap_case


@wrap_case
class ClearStartEndDashWrapTest(TestCase):

    @wrap_case('-my string', 'my string')
    @wrap_case('-  -- --my string', 'my string')
    @wrap_case('my string-', 'my string')
    @wrap_case('my string-- -- -- - ', 'my string')
    @wrap_case('my-string', 'my-string')
    def test_remove_dash(self, string, expected_result):
        result = clear_start_end_dash(string)
        self.assertEqual(result, expected_result)
```

This code generates 5 tests, that works like a simple test functions.

Console output:

```
test_remove_dash_u'-  -- --my string'_u'my string' (tests.example.test_simple_test.ClearStartEndDashW
test_remove_dash_u'-my string'_u'my string' (tests.example.test_simple_test.ClearStartEndDashWrapTest
test_remove_dash_u'my string-'_u'my string' (tests.example.test_simple_test.ClearStartEndDashWrapTest
test_remove_dash_u'my string-- -- -- - '_u'my string' (tests.example.test_simple_test.ClearStartEndDa
test_remove_dash_u'my-string'_u'my-string' (tests.example.test_simple_test.ClearStartEndDashWrapTest)
```

1.2 Installation

```
pip install python_wrap_cases
```

Free software: BSD license

Documentation: https://python_wrap_cases.readthedocs.org.

CHAPTER 2

Installation

At the command line:

```
$ easy_install python_wrap_cases
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv python_wrap_cases
$ pip install python_wrap_cases
```

CHAPTER 3

Usage

To use Python wrap cases in a project

```
from unittest import TestCase
from python_wrap_cases import wrap_case

@wrap_case
class SomeTest(TestCase):

    @wrap_case('value1_a', 'value2_a')
    @wrap_case('value1_b', 'value2_b')
    def test_with_params(self, param1, param2)
        # ...
```

Just add mixin WrapCasesMixin to your test class and add decorators @wrap_case to test function with parameters that you wanna add to test method.

3.1 Usage with mock

By default wrap_case detects the mock arguments and changes a return_value.

```
import unittest
from python_wrap_cases import wrap_case
from mock import patch


class TestedClass():

    def foo_a(self):
        pass

    def foo_b(self):
        pass

    def tested_method(self):
        return self.foo_a() + self.foo_b()

@wrap_case
class TestsWithMock(unittest.TestCase):

    @wrap_case(2, 2, 4)
    @wrap_case(3, 3, 6)
```

```
    @wrap_case(4, 4, 8)
    @patch.object(TestedClass, 'foo_b')
    @patch.object(TestedClass, 'foo_a')
    def test_with_patch_object(self, return_value):
        tested_class = TestedClass()
        self.assertEqual(tested_class.tested_method(), return_value)

    @wrap_case(foo_a=2, result=4)
    @wrap_case(foo_a=4, result=6)
    @wrap_case(foo_a=8, result=10)
    @patch.multiple(TestedClass, foo_a=DEFAULT, foo_b=DEFAULT)
    def test_with_patch_multiple(self, result, foo_b):
        foo_b.return_value = 2
        tested_class = TestedClass()
        self.assertEqual(tested_class.tested_method(), result)
```

3.2 ListGenerator

List generator helps to generate cases based on list of arguments.

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
class ListGeneratorTests(unittest.TestCase):

    @wrap_case(number__list=[0, 1, 2, 3])
    def test_div_1(self, number):
        self.assertEqual(number/1, number)
```

This code will work like this one:

List generator helps to generate cases based on list of arguments.

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
class TestsWithoutListGenerator(unittest.TestCase):

    @wrap_case(number=0)
    @wrap_case(number=1)
    @wrap_case(number=2)
    @wrap_case(number=3)
    def test_div_1(self, number):
        self.assertEqual(number/1, number)
```

If you use two or more list generator in wrap_case, library will generate all possible combination of arguments from these lists.

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
```

```
class TestWithTwoListGenerators(unittest.TestCase):

    @wrap_case(a__list=[1, 2], b__list=[0, 1])
    def test_gte(self, a, b):
        self.assertTrue(a >= b)
```

it's equal to:

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
class TestWithoutListGenerators(unittest.TestCase):

    @wrap_case(a=1, b=0)
    @wrap_case(a=1, b=1)
    @wrap_case(a=2, b=0)
    @wrap_case(a=2, b=1)
    def test_gte(self, a, b):
        self.assertTrue(a >= b)
```

3.3 SyncListGenerator

The same as ListGenerator but instead of generate all possible argument combination it generate cases successively.

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
class TestWithSyncListGenerator(unittest.TestCase):

    @wrap_case(number__sync_list=[0, 1, 2, 3], result__sync_list=[1, 2, 3, 4])
    def test_add_1(self, number, result):
        self.assertEqual(number + 1, result)
```

it's equal to:

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
class TestWithoutSyncListGenerator(unittest.TestCase):

    @wrap_case(number=0, result=1)
    @wrap_case(number=1, result=2)
    @wrap_case(number=2, result=3)
    @wrap_case(number=3, result=4)
    def test_add_1(self, number, result):
        self.assertEqual(number + 1, result)
```

3.4 CustomGenerator

If you need more flexible generator you may use CustomGenerator

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
class CustomGenerators(unittest.TestCase):

    @wrap_case(number__list=[0, 1, 2, 3], result__custom=lambda number, result: number + 1)
    def test_add_1(self, number, result):
        self.assertEqual(number + 1, result)
```

3.5 FuncGenerator

Simple as a CustomGenerator but without arguments.

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
class FuncGenerator(unittest.TestCase):

    @wrap_case(string__func=lambda: 'Hello World{0}'.format('!'*3))
    def test_simple_func(self, string):
        self.assertEqual(string, 'Hello World!!!')
```

3.6 RangeGenerator

Generate range of numbers

```
import unittest
from python_wrap_cases import wrap_case


@wrap_case
class RangeGenerator(unittest.TestCase):

    @wrap_case(number__range=4)
    def test_range_4_div_1(self, number):
        self.assertEqual(number/1, number)

    @wrap_case(number__range=(1, 4, ))
    def test_range_1_4_div_1(self, number):
        self.assertEqual(number/1, number)

    @wrap_case(number__range=(1, 4, 2, ))
    def test_range_1_4_2_div_1(self, number):
        self.assertEqual(number/1, number)
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/erm0l0v/python_wrap_cases/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

4.1.4 Write Documentation

Python wrap cases could always use more documentation, whether as part of the official Python wrap cases docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/erm0l0v/python_wrap_cases/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *python_wrap_cases* for local development.

1. Fork the *python_wrap_cases* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/python_wrap_cases.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv python_wrap_cases
$ cd python_wrap_cases/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 python_wrap_cases tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/erm0l0v/python_wrap_cases/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_python_wrap_cases
```

CHAPTER 5

Credits

5.1 Development Lead

- Kirill Ermolov <erm0l0v@ya.ru>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

CHAPTER 7

0.1.0 (2015-06-26)

- First release on PyPI.

0.1.2 (2015-06-26)

- Fix generators import

CHAPTER 9

0.1.3 (2015-06-26)

- Add some docs

CHAPTER 10

0.1.4 (2015-06-29)

- ReadMe add semicolon
- Fix pypi readme

CHAPTER 11

0.1.5 (2015-07-01)

- README remove :::;

CHAPTER 12

0.1.6 (2015-07-01)

- Add tests fot python 3.2

0.1.7 (2015-07-10)

- Add six dependency

CHAPTER 14

0.1.8 (2015-08-21)

- Add func generator
- Add range generator
- Fix problem with iterator in custom generator
- Add new API for declaration wrapped TestCase. (added wrap_case decorator without parameters)

CHAPTER 15

Indices and tables

- genindex
- modindex
- search