pytest framework — intro & overview

Литус Ярослав – группа тестирования поиска

ООО «ПП Спутник»

28 апреля 2016 г.

План

1. unittest vs. pytest
2. assert-выражения и отчет об ошибках
3. Параметризация
4. Выполнение тестов
5. Маркеры
6. Немного о фикстурах
7. Плагины для pytest
8. Полезные ресурсы

Unittest code example

```
1  import unittest
2
3  class TestStringMethods(unittest.TestCase):
4
5    def test_isupper(self):
6        self.assertTrue('FOO'.isupper())
7        self.assertFalse('Foo'.isupper())
8
9    def test_split(self):
10       s = 'hello world'
11       self.assertEqual(s.split(), ['hello', 'world'])
12       with self.assertRaises(TypeError):
13           s.split(2)
14
15 if __name__ == '__main__':
16     unittest.main()
```

Same thing done with pytest

```
1  import pytest
2
3  def test_isupper():
4      assert "FOO".isupper()
5      assert not "Foo".isupper()
6
7  def test_split():
8      s = 'hello world'
9      assert s.split() == ['hello', 'world']
10     with pytest.raises(TypeError):
11         s.split(2)
```

Unittest:

```
Traceback (most recent call last):
  File "code/unittest-example.py", line 11, in test_split
    self.assertEqual(s.split(), ['hello'])
AssertionError: Lists differ: ['hello', 'world'] != ['hello']

First list contains 1 additional elements.
First extra element 1:
world

- ['hello', 'world']
+ ['hello']
```

Unittest:

```
Traceback (most recent call last):
  File "code/unittest-example.py", line 11, in test_split
    self.assertEqual(s.split(), ['hello'])
AssertionError: Lists differ: ['hello', 'world'] != ['hello']

First list contains 1 additional elements.
First extra element 1:
world

- ['hello', 'world']
+ ['hello']
```

Pytest:

```
____________________________________ test_split ____________________________________

    def test_split():
        s = 'hello world'
>       assert s.split() == ['hello']
E       assert ['hello', 'world'] == ['hello']
E         Left contains more items, first extra item: 'world'
E         Use -v to get the full diff
```

- Похоже, не правда ли?

- Похоже, не правда ли?
- Но как это достигается?

Assertions and Error reporting

- Похоже, не правда ли?
- Но как это достигается?
- Unittest использует специальные assert-методы класса TestCase.

Assertions and Error reporting

- Похоже, не правда ли?
- Но как это достигается?
- Unittest использует специальные assert-методы класса TestCase.
- Pytest переопределяет assert-выражения при подгрузке (import) модуля.

Assertions and Error reporting

- Похоже, не правда ли?
- Но как это достигается?
- Unittest использует специальные assert-методы класса TestCase.
- Pytest переопределяет assert-выражения при подгрузке (import) модуля.

> *pytest has support for showing the values of the most common subexpressions including calls, attributes, comparisons, and binary and unary operators. (See Demo of Python failure reports with pytest). This allows you to use the idiomatic python constructs without boilerplate code while not losing introspection information.*

Assert-методы в unittest

Самые распространенные assert-методы класса TestCase:

```
assertEqual(a, b)
assertNotEqual(a, b)
assertTrue(x)
assertFalse(x)
assertIs(a, b)
assertIsNot(a, b)
assertIsNone(x)
assertIsNotNone(x)
assertIn(a, b)
assertNotIn(a, b)
assertIsInstance(a, b)
assertNotIsInstance(a, b)
```

Test parametrization

- Рассмотрим ситуацию когда требуется выполнить один и тот же сценарий с разными данными.

Test parametrization

- Рассмотрим ситуацию когда требуется выполнить один и тот же сценарий с разными данными.
- К примеру требуется протестировать строковый метод `isupper()` на разных строках.

Test parametrization

- Рассмотрим ситуацию когда требуется выполнить один и тот же сценарий с разными данными.
- К примеру требуется протестировать строковый метод `isupper()` на разных строках.
- Unittest не предоставляет инструмента для параметризации тестов, — каждый кейс придется описывать вручную, что приведет к множественному дублированию кода и существенно затруднит его поддержку.

Test parametrization

- Рассмотрим ситуацию когда требуется выполнить один и тот же сценарий с разными данными.
- К примеру требуется протестировать строковый метод `isupper()` на разных строках.
- Unittest не предоставляет инструмента для параметризации тестов, — каждый кейс придется описывать вручную, что приведет к множественному дублированию кода и существенно затруднит его поддержку.
- Что же скажет pytest?

Test parametrization

- Рассмотрим ситуацию когда требуется выполнить один и тот же сценарий с разными данными.
- К примеру требуется протестировать строковый метод `isupper()` на разных строках.
- Unittest не предоставляет инструмента для параметризации тестов, — каждый кейс придется описывать вручную, что приведет к множественному дублированию кода и существенно затруднит его поддержку.
- Что же скажет pytest?
- Удобная параметризация из коробки!

Test parametrization — example

```
1  #!coding: utf-8
2  import pytest
3
4  @pytest.mark.parametrize(
5      "string,is_upper",
6      (
7          ("UPPER", True),
8          ("lower", False),
9          ("CamelCase", False),
10         (u"СПУТНИК", True)
11     )
12 )
13 def test_isupper(string, is_upper):
14     assert string.isupper() == is_upper
```

- Пример запуска тестов:

```
py.test dir_1 ../tests dir_2/test_this.py
```

- Пример запуска тестов:

  ```
  py.test dir_1 ../tests dir_2/test_this.py
  ```

- If no arguments are specified then collection starts from testpaths (if configured) or the current directory. Alternatively, command line arguments can be used in any combination of directories, file names or node ids;

- Пример запуска тестов:

  ```
  py.test dir_1 ../tests dir_2/test_this.py
  ```

- If no arguments are specified then collection starts from testpaths (if configured) or the current directory. Alternatively, command line arguments can be used in any combination of directories, file names or node ids;
- recurse into directories, unless they match *norecursedirs*;

- Пример запуска тестов:

  ```
  py.test dir_1 ../tests dir_2/test_this.py
  ```

- If no arguments are specified then collection starts from testpaths (if configured) or the current directory. Alternatively, command line arguments can be used in any combination of directories, file names or node ids;
- recurse into directories, unless they match *norecursedirs*;
- `test_*.py` or `*_test.py` files, imported by their test package name;

Markers & test collection

- Пример запуска тестов:

  `py.test dir_1 ../tests dir_2/test_this.py`
- If no arguments are specified then collection starts from testpaths (if configured) or the current directory. Alternatively, command line arguments can be used in any combination of directories, file names or node ids;
- recurse into directories, unless they match *norecursedirs*;
- `test_*.py` or `*_test.py` files, imported by their test package name;
- Test prefixed test classes (without an `__init__` method);

Markers & test collection

- Пример запуска тестов:

  `py.test dir_1 ../tests dir_2/test_this.py`
- If no arguments are specified then collection starts from testpaths (if configured) or the current directory. Alternatively, command line arguments can be used in any combination of directories, file names or node ids;
- recurse into directories, unless they match *norecursedirs*;
- `test_*.py` or `*_test.py` files, imported by their test package name;
- Test prefixed test classes (without an `__init__` method);
- `test_` prefixed test functions or methods are test items.

Markers & test collection

- Механизм маркеров дает удобный способ организации тестов и их выборочного запуска.

Markers & test collection

- Механизм маркеров дает удобный способ организации тестов и их выборочного запуска.
- Как пометить тест(ы) маркером?

Markers & test collection

- Механизм маркеров дает удобный способ организации тестов и их выборочного запуска.
- Как пометить тест(ы) маркером?
-
```
@pytest.mark.smoke
def test_do_this_and_that():
    # ...

@pytest.mark.smoke
@pytest.mark.slow
def test_do_smth_very_slow():
    # ...
```

Markers & test collection

- Механизм маркеров дает удобный способ организации тестов и их выборочного запуска.
- Как пометить тест(ы) маркером?
-
```
@pytest.mark.smoke
def test_do_this_and_that():
    # ...

@pytest.mark.smoke
@pytest.mark.slow
def test_do_smth_very_slow():
    # ...
```
- Помечая тесты таким образом, мы получаем способ гибко формировать комплекты для запуска:

```
py.test tests/ -m smoke
```

```
py.test tests/ -m "smoke and not slow"
```

Markers & test collection

- Есть несколько полезных встроенных маркеров.

Markers & test collection

- Есть несколько полезных встроенных маркеров.
- `py.test --markers`

Markers & test collection

- Есть несколько полезных встроенных маркеров.
- `py.test --markers`
- `@pytest.mark.skipif(condition)`

  Example: `skipif('sys.platform == "win32"')` skips the test if we are on the win32 platform.

Markers & test collection

- Есть несколько полезных встроенных маркеров.
- `py.test --markers`
- `@pytest.mark.skipif(condition)`

  Example: `skipif('sys.platform == "win32"')` skips the test if we are on the win32 platform.
- `@pytest.mark.xfail(condition, reason, run, raises)`

  See http://pytest.org/latest/skipping.html

Markers & test collection

- Есть несколько полезных встроенных маркеров.
- `py.test --markers`
- `@pytest.mark.skipif(condition)`

  Example: `skipif('sys.platform == "win32"')` skips the test if we are on the win32 platform.
- `@pytest.mark.xfail(condition, reason, run, raises)`

  See `http://pytest.org/latest/skipping.html`
- `@pytest.mark.parametrize(argnames, argvalues)`

  Example: `@parametrize('arg1', [1,2])` would lead to two calls of the decorated test function, one with arg1=1 and another with arg1=2

Немного о фикстурах

- Одним из самых удачных решений фреймворка pytest стали фикстуры (fixtures).

Немного о фикстурах

- Одним из самых удачных решений фреймворка pytest стали фикстуры (fixtures).
- In software testing, a test fixture is a fixed state of the software under test used as a baseline for running tests; also known as the test context.

Немного о фикстурах

- Одним из самых удачных решений фреймворка pytest стали фикстуры (fixtures).
- In software testing, a test fixture is a fixed state of the software under test used as a baseline for running tests; also known as the test context.
- В классических xUnit-фреймворках фиксуры реализуются через описание действий, относящихся к setUp и к tearDown. Например в unittest необходимо переопределить методы setUp() и к tearDown() класса TestCase.

Немного о фикстурах

- Одним из самых удачных решений фреймворка pytest стали фикстуры (fixtures).
- In software testing, a test fixture is a fixed state of the software under test used as a baseline for running tests; also known as the test context.
- В классических xUnit-фреймворках фиксуры реализуются через описание действий, относящихся к setUp и к tearDown. Например в unittest необходимо переопределить методы setUp() и к tearDown() класса TestCase.
- Pytest вводит понятие функции-фикстуры, которая передается в тест в виде funcarg-а.

Немного о фикстурах

- Одним из самых удачных решений фреймворка pytest стали фикстуры (fixtures).
- In software testing, a test fixture is a fixed state of the software under test used as a baseline for running tests; also known as the test context.
- В классических xUnit-фреймворках фиксуры реализуются через описание действий, относящихся к setUp и к tearDown. Например в unittest необходимо переопределить методы setUp() и к tearDown() класса TestCase.
- Pytest вводит понятие функции-фикстуры, которая передается в тест в виде funcarg-а.
- ЧЕГО???

Немного о фикстурах

- Одним из самых удачных решений фреймворка pytest стали фикстуры (fixtures).
- In software testing, a test fixture is a fixed state of the software under test used as a baseline for running tests; also known as the test context.
- В классических xUnit-фреймворках фиксуры реализуются через описание действий, относящихся к setUp и к tearDown. Например в unittest необходимо переопределить методы setUp() и к tearDown() класса TestCase.
- Pytest вводит понятие функции-фикстуры, которая передается в тест в виде funcarg-а.
- ЧЕГО???
- О фикстурах стоит поговоить отдельно и поподробнее. . .

Extensions/plugins

- pytest является во многих смыслах гибким фреймворком.

Extensions/plugins

- pytest является во многих смыслах гибким фреймворком.
- Подробная документация и богатое API дают возможность достаточно просто писать плагины.

Extensions/plugins

- pytest является во многих смыслах гибким фреймворком.
- Подробная документация и богатое API дают возможность достаточно просто писать плагины.
- В сообществе уже существует обширный набор: около 200 плагинов

Extensions/plugins

- pytest является во многих смыслах гибким фреймворком.
- Подробная документация и богатое API дают возможность достаточно просто писать плагины.
- В сообществе уже существует обширный набор: около 200 плагинов
- Установка — как для обычного пакета:

```
pip install pytest-*
```

Extensions/plugins

- pytest является во многих смыслах гибким фреймворком.
- Подробная документация и богатое API дают возможность достаточно просто писать плагины.
- В сообществе уже существует обширный набор: около 200 плагинов
- Установка — как для обычного пакета:

  ```
  pip install pytest-*
  ```

- Как подойти к написанию своего плагина?

Extensions/plugins

- pytest является во многих смыслах гибким фреймворком.
- Подробная документация и богатое API дают возможность достаточно просто писать плагины.
- В сообществе уже существует обширный набор: около 200 плагинов
- Установка — как для обычного пакета:

  ```
  pip install pytest-*
  ```

- Как подойти к написанию своего плагина?
- Все что потребуется — изучить спецификации предоставляемых хуков!

pytest hooks

```
# Initialization, command line and configuration hooks:
pytest_load_initial_conftests
pytest_cmdline_preparse(config, args)
pytest_cmdline_parse(pluginmanager, args)
pytest_namespace()
pytest_addoption(parser)
pytest_cmdline_main(config)
pytest_configure(config)
pytest_unconfigure(config)

# Collection hooks:
pytest_ignore_collect(path, config)
pytest_collect_directory(path, parent)
pytest_collect_file(path, parent)
pytest_pycollect_makeitem(collector, name, obj)
pytest_generate_tests(metafunc)
pytest_collection_modifyitems(session, config, items)
```

pytest hooks

```
# Reporting hooks:
pytest_collectstart(collector)
pytest_itemcollected(item)
pytest_collectreport(report)
pytest_deselected(items)
pytest_report_header(config, startdir)
pytest_report_teststatus(report)
pytest_terminal_summary(terminalreporter)
pytest_runtest_logreport(report)
pytest_assertrepr_compare(config, op, left, right)

# Debugging/Interaction hooks:
pytest_internalerror(excrepr, excinfo)
pytest_keyboard_interrupt(excinfo)
pytest_exception_interact(node, call, report)
pytest_enter_pdb(config)
```

Plugins are easy!

Синтетический пример (из другой презентации).

```
# conftest.py
def pytest_addoption(parser):
    parser.addoption(
        "--backend",
        choices=("webkit", "webengine"),
        default="webkit"
    )

# test_foo.py
def test_foo(request):
    if request.config.getoption("--backend") == "webkit":
        # ...
    else:
        # ...
```

Написанию плагинов стоит посвятить отдельный доклад.

to be continued...

Полезные ресурсы:

- Документация и примеры (Ваш ресурс номер 1): http://pytest.org. Там же коллекция ссылок на доклады и блоги: http://pytest.org/latest/talks.html
- Список плагинов: http://plugincompat.herokuapp.com
- Вопросы-ответы на stackoverflow: http://stackoverflow.com/search?q=pytest
- Интересная перезентация на русском: http://www.slideshare.net/imankulov/pytest-testing
- Интересная перезентация на английском: http://www.the-compiler.org/tmp/sps16.html
- Создатель фреймворка: https://holgerkrekel.net/

При создании данной презентации были использованы материалы некоторых из перечисленных ресурсов