Памятка по написанию тестов при помощи PyTest

Источник: https://eax.me/pytest/

Допустим, разрабатывается некоторый проект. К проекту требуется написать интеграционные и системные тесты, а также, возможно, нагрузочные и еще какие-то. Для решения этой задачи [Python](https://eax.me/python/) подходит просто идеально. В чем мы с вами скоро и убедимся, познакомившись с фреймворком PyTest и некоторыми плагинами к нему.

Установка:

sudo pip install pytest pytest-quickcheck pytest-html pytest-cov

Чтобы было что тестировать, создадим незамысловатый модуль fibgen, вся реализация которого приведена ниже:

def fibgen(num):  
    """  
    Generates Fibonacci numbers  
  
    :param num: how many numbers to generate  
    :return: generator of first num Fibonacci numbers  
  
    >>> list(fibgen(10))  
    [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]  
    """  
    if type(num) is not int:  
        raise TypeError("type of num argument should be int")  
    if num <= 0:  
        return  
    x1 = 0  
    x2 = 1  
    i = 1  
    yield 1  
    while i < num:  
        i = i + 1  
        res = x1 + x2  
        x1 = x2  
        x2 = res  
        yield res  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    print(list(fibgen(10)))

Файл tests/test\_fibgen.py:

from fibgen import fibgen  
import pytest  
  
class TestFibgen:  
  
    def test\_type\_error(self):  
        with pytest.raises(TypeError):  
            list(fibgen('ololo'))  
  
    def test\_negative(self):  
        assert(list(fibgen(-1)) == [])  
  
    def test\_empty(self):  
        assert(list(fibgen(0)) == [])  
  
    def test\_one(self):  
        assert(list(fibgen(1)) == [1])  
  
    def test\_two(self):  
        assert(list(fibgen(2)) == [1,1])  
  
    def test\_three(self):  
        assert(list(fibgen(3)) == [1, 1, 2])  
  
    def test\_seven(self):  
        *# result = list(fibgen(10))*  
        result = list(fibgen(7))  
        expected = [1, 1, 2, 3, 5, 8, 13]  
        assert(result == expected)  
  
    @pytest.mark.randomize(num=int, min\_num=3, max\_num=1000, ncalls=99)  
    def test\_quickcheck(self, num):  
        result = list(fibgen(num))  
        assert(result[0] < result[-1])  
        assert(len(result) == num)

Чтобы было несколько тестовых наборов, создадим еще tests/subdir/test\_math.py:

class TestMath:  
  
    def setup\_class(self):  
        print("\n=== TestMath - setup class ===\n")  
  
    def teardown\_class(self):  
        print("\n=== TestMath - teardown class ===\n")  
  
    def setup(self):  
        print("TestMath - setup method")  
  
    def teardown(self):  
        print("TestMath - teardown method")  
  
    def test\_add(self):  
        assert(2 + 2 == 4)  
  
    def test\_mul(self):  
        assert(3 \* 3 == 9)

Теперь попробуем разобраться, что же мы здесь видим.

Фреймворк PyTest находит тесты, глядя на имена файлов, классов и методов. Файлы с тестами должны содержать в имени строку «test\_», имена классов — начинаться с «Test», а методов — со строки «test\_». Все проверки почти всегда выполняются при помощи assert, то есть, никаких навороченных DSL, как в [ScalaTest](https://eax.me/scalatest/), не предусмотрено. Если требуется проверить, что код бросает некое исключение, можно использовать конструкцию with pytest.raises, как в тесте test\_type\_error.

Вдумчивые читатели могли заподозрить [property-based тест](https://eax.me/scalacheck/) в коде:

*# ...*  
    @pytest.mark.randomize(num=int, min\_num=3, max\_num=1000, ncalls=99)  
    def test\_quickcheck(self, num):  
        result = list(fibgen(num))  
        assert(result[0] < result[-1])  
        assert(len(result) == num)  
*# ...*

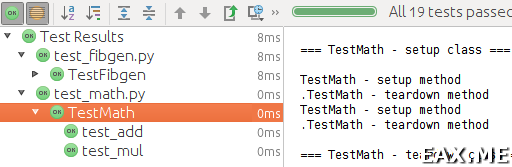
Все правильно, это он и есть. Аргумент num будет генерироваться случайным образом из диапазона 3..1000, всего же тест будет запущен 99 раз. Следует однако иметь ввиду, что property-based тесты в PyTest *намного* тупее, нежели в QuickCheck или ScalaCheck. Так, к примеру, никакой гарантированной проверки граничных случаев нет, тест просто выполняется с какими-то случайными значениями. Кроме того, shrinking не поддерживается.

Следующий отрывок кода говорит сам за себя (так называемые «фикстуры»):

*# ...*  
    def setup\_class(self):  
        print("\n=== TestMath - setup class ===\n")  
  
    def teardown\_class(self):  
        print("\n=== TestMath - teardown class ===\n")  
  
    def setup(self):  
        print("TestMath - setup method")  
  
    def teardown(self):  
        print("TestMath - teardown method")  
*# ...*

Метод с именем setup выполняется перед каждым тестом, а с именем teardown, вполне закономерно — после. Аналогично метод setup\_class выполняется перед началом выполнения всех тестов в классе. После того, как все тесты в классе были выполнены, вызывается teardown\_class.

Самый простой способ запустить все тесты — воспользоваться [PyCharm](https://www.jetbrains.com/pycharm/):



Запуск всех тестов из консоли:

PYTHONPATH=. py.test

Переменная окружения PYTHONPATH нужна для того, чтобы PyTest нашел пакет fibgen. Если вы пишите интеграционные или системные тесты, они вполне могут опираться на модули, объявленные в текущем каталоге. Так что, эту переменную лучше всегда выставлять, чтобы не наткнуться на ошибку вроде ImportError: No module named fibgen.

Желательно всегда использовать verbose режим, чтобы видеть, какие тесты были запущены, видеть полностью какие assert’ы и почему провалились, и так далее:

PYTHONPATH=. py.test -v

Флаг -l включает отображение значений переменных при падении теста:

PYTHONPATH=. py.test -v -l

Запуск конкретного набора тестов, короткая версия:

PYTHONPATH=. py.test -v -l -k TestMath  
PYTHONPATH=. py.test -v -l -k TestFibgen

Запуск тестов из конкретного файла:

PYTHONPATH=. py.test -v -l tests/subdir/test\_math.py

Из конкретного класса:

PYTHONPATH=. py.test -v -l tests/subdir/test\_math.py::TestMath

Запуск конкретного метода:

PYTHONPATH=. py.test -v -l \  
  tests/subdir/test\_math.py::TestMath::test\_add

Можно указывать несколько тестов:

PYTHONPATH=. py.test -v -l \  
  tests/subdir/test\_math.py::TestMath::test\_add \  
  tests/test\_fibgen.py::TestFibgen::test\_empty

Тестирование примеров кода из документации (doctest):

PYTHONPATH=. py.test --doctest-modules -v -l

Определение степени покрытия модуля fibgen тестами с выводом номеров непокрытых строк в терминал и генерацией отчета о покрытии в HTML:

PYTHONPATH=. py.test -v -l --cov=fibgen --cov-report term-missing \  
  --cov-report html

Кстати, оказывается, что в PyCharm Community Edition [нет поддержки code coverage](http://blog.jetbrains.com/pycharm/2015/06/feature-spotlight-python-code-coverage-with-pycharm/" \l "comment-257192), этот функционал есть только в Professional Edition.

C генерацией junit.xml:

PYTHONPATH=. py.test -v -l --junit-xml=junit.xml

С генерацией HTML:

PYTHONPATH=. py.test -v -l --html=out.html

К сожалению, приведенная команда генерирует один большой HTML-файл на все наборы тестов. Решить эту проблему можно командой:

PYTHONPATH=. find tests -type f -iname '\*.py' \  
  -exec py.test -v -l {} --html=html/{}.html \;

Но, увы, в этом случае мы потеряем коды возврата, что плохо работает с [системами непрерывной интеграции вроде Jenkins](https://eax.me/jenkins/).

Поэтому я в итоге написал такую обвязку:

*#!/usr/bin/env python*  
  
import os  
import re  
import subprocess  
  
html\_prefix = "./html/"  
idx\_html = "<table>\n<tr><td>test</td><td>status</td></tr>\n"  
  
for root, dirs, files in os.walk("tests"):  
    for fname in files:  
        if re.search("\\.py$", fname) is not None:  
            sname = os.path.join(root, fname)  
            html = html\_prefix + sname + ".html"  
            cmd = "PYTHONPATH=. py.test --doctest-modules -v -l " + \  
                  sname + " --html=" + html  
            code = subprocess.call(cmd, shell = True)  
            status = "<p style='color:green;'>Success</p>"  
            if code != 0:  
                status = "<p style='color:red;'>FAILURE</p>"  
            idx\_html = idx\_html + "<tr><td><a href='" + sname + \  
                       ".html'>" + sname + "</a></td><td>" + status + \  
                       "</td></tr>\n"  
  
idx\_html = idx\_html + "</table>\n"  
  
with open(html\_prefix + "index.html", "w") as f:  
    f.write(idx\_html)

Ccылки по теме:

* [Официальный сайт PyTest](https://pytest.org/);
* Модули на PyPI: [pytest-quickcheck](https://pypi.python.org/pypi/pytest-quickcheck), [pytest-html](https://pypi.python.org/pypi/pytest-html), [pytest-cov](https://pypi.python.org/pypi/pytest-cov);
* [Еще больше примеров использования фикстур (fixtures) в PyTest](http://pythontesting.net/framework/pytest/pytest-introduction/#fixtures);
* [Тут чуть больше подробностей про конструкцию pytest.raises](https://pytest.org/latest/assert.html#assertions-about-expected-exceptions);
* [Allure](http://allure.qatools.ru/) — говорят, [умеет делать](http://stackoverflow.com/a/27175962/1565238) красивые репорты из junit.xml;
* [Полная версия исходников к этому посту на GitHub](https://github.com/afiskon/pytest-example);