pspecs Documentation

Release 0.1.5

Catalin Costache

Sep 27, 2017

Contents

pspecs is a testing library that concentrates on making it easy to build test subjects, isolate test runs and write readable specifications

It has very few conventions and it provides integrations with established python testing tools like py.test or nose

Contents:

CHAPTER 1

Quickstart

Installation

```
>>> pip install pspecs
```

Basic structure

```
# my_first_spec.py
from pspecs import Context, let

class DescribeMath(Context):

    @let
    def numbers(self):
        return [1, 2, 3]

    class DescribeSum(Context):

        @let
        def sum(self):
            return sum(self.numbers)

        def it_should_be_six(self):
            assert self.sum == 6

    class DescribeProd(Context):

        @let
        def prod(self):
            return reduce(lambda a, b: a*b, self.numbers, 1)
```

```
        def it_should_be_six(self):
            assert self.prod == 6
```

It should be fairly easily to spot what's being tested here: the sum and the product of a sequence of integers.

You will notice few off the simple and powerful concepts in *pspecs*: Memoized let methods and Parent and children contexts.

Running

pspecs doesn't provide it's own runner yet. Instead, it relies on well known test runners like py.test or nose

Running with nose:

```
>>> nosetests --with-specs my_first_spec.py
```

Tests narratives

Given a spec file **order_spec.py**

```
from pspecs import Context

class DescribeOrder(Context):

    def let_quantity(self): return 10
    def let_money(self):
        return Money(10, self.type)

    def let_type(self):
        return 'USD'

    def let_order(self):
        return Order(self.quantity, self.money)

    class DescribeOrderTotal(Context):
        def let_total(self):
            return self.order.total()

        def it_should_account_for_quantity(self):
            assert self.total == 100

        class WithEmptyQuantity(Context):
            def let_quantity(self): return 0

            def it_should_be_zero(self):
                assert self.total == 0

    class DescribeOrderCurrency(Context):
        def let_currency(self):
            return self.order.currency()

        def it_should_be_dollars(self):
            assert self.currency == 'USD'

```

```
35          class WithEuros(Context):
36
37              def it_should_be_euros(self):
38                  self.type = 'EUR'
39                  assert self.currency == 'EUR'
```

The purpose of any testing library is to make it easy for anybody looking at it to spot quickly what's being tested.

Looking at the code above and reading only the class names and the *it_* method names, we can discover a narrative:

```
DescribeOrder
    DescribeOrderTotal
        it_should_account_for_quantity
        WithEmptyContext
            it_should_be_zero
    DescribeOrderCurrency
        it_should_be_dollars
        WithEuros
            it_should_be_euros
```

What *pspec* does is exactly to allow such kind of narratives to be built easily using simple, yet powerful concepts like *contexts*, *memoized let methods*, *hooks* and *isolated tests*

CHAPTER 2

Indices and tables

- genindex
- modindex
- search