Allows Documentation

Release 0.1.0

Dave Anderson

Jun 01, 2019

Contents:

CHAPTER 1

Allows

Easier mock configuration and assertions in Python using R-spec-like grammar!

```
allow(my_mock).to(return_value('hi').on_method('wave'))
allow(my_mock).to(return_value('bye').on_method('wave').when_called_with('see ya'))

assert my_mock.wave() == 'hi'
assert my_mock.wave('see ya') == 'bye'
```

This library is built to wrap and configure Mock, MagicMock and other objects from the built in unittest.mock available in Python 3.3+.

- Free software: MIT license
- Documentation: https://allows.readthedocs.io.

1.1 Features

- R-spec-like grammar for specifing Mock behavior
- Compatible with all Python standard library unittest.mock Mock (MagicMock, Patch, etc.)
- Stand alone SideEffect builder to model and combine complex side effects

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Allows, run this command in your terminal:

```
$ pip install allows
```

This is the preferred method to install Allows, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Allows can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/dvndrsn/allows
```

Or download the tarball:

```
$ curl  -OL https://github.com/dvndrsn/allows/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Allows in a project:

```
from unittest.mock import Mock

from allows import allow, receive_method

my_mock = Mock()
allow(my_mock).to(
    receive_method('spam')
    .called_with(eggs='foo')
    .and_return_value('bar')
)
allow(my_mock).to(
    receive_method('spam')
    .called_with(eggs='no thanks')
    .and_return_value('ok')
)

assert my_mock.spam(eggs='foo') == 'bar'
assert my_mock.spam(eggs='no thanks') == 'ok'
```

Allows specifies a similar grammar to Rspec to set up Mocks for tests. This allows complex side effects to be constructed more easily than with the standard Mock API.

Allows can be used without the grammar to build and compose side effects as well.

```
from unittest.mock import Mock, call

from allows import SideEffect, SideEffectBuilder

side_effect = SideEffectBuilder() \
    .with_return_value('bar') \
    .with_call_args(eggs='foo')
```

(continues on next page)

(continued from previous page)

```
args = call(eggs='no thanks')
effect = lambda *args, **kwargs: 'bar'
another_side_effect = SideEffect(args, effect)

side_effect.merge(another_side_effect)

my_mock = Mock()
my_mock.spam.side_effect = side_effect

assert my_mock.spam(eggs='foo') == 'bar'
assert my_mock.spam(eggs='no thanks') == 'ok'
```

CHAPTER 4

allows

4.1 allows package

allows.**allow**(*mock_subject: unittest.mock.Mock*) → allows.grammar.MockExtensionGrammar
Prepare to extend a Mock from the Python Standard Library with a SideEffect.

allows.**receive_method**(*name: str*) → allows.grammar.SideEffectBuilderGrammar
Start building a side effect on a named method of a Mock.

```
my_mock = Mock()
allow(my_mock).to(receive_method('foo').and_return('bar'))

assert my_mock.foo() == 'bar'
```

allows.**return_value**(*return_value: Any*) → allows.grammar.SideEffectBuilderGrammar
Start building a side effect which returns a value when called.

```
my_mock = Mock()
allow(my_mock).to(return_value('fooby'))

assert my_mock() == 'fooby'
```

allows.**raise_exception**(*raised_exception: Exception*) → allows.grammar.SideEffectBuilderGrammar
Start building a side effect which raises an exception.

```
my_mock = Mock()
allow(my_mock).to(raise_exception(ValueError))

raised = False
try:
    my_mock()
except ValueError:
    raised = True
assert raised
```

allows.**be_called_with**(**args*, ***kwargs*) → allows.grammar.SideEffectBuilderGrammar
Start building a side effect which accepts arguments and keyword arguments

```
my_mock = Mock()
allow(my_mock).to(be_called_with('spam', foo='bar').and_return('eggs))

assert my_mock('spam', foo='bar') == 'eggs'
```

4.1.1 Submodules

4.1.2 allows.grammar module

class allows.grammar.**MockExtensionGrammar**(*mock_subject: unittest.mock.Mock*)
MockExtensionGrammar is created by the allow factory.

This enables grammar for creating and binding a mock side effect like:

allow **<Mock>** *to* **<Have Side Effect>**

class allows.grammar.**SideEffectBuilderGrammar**(*method_name=None*, *builder=None*)
SideEffectBuilderGrammar is initiated by the return_value, raise_exception, receive_method, be_called_with, have_effect factory methods.

The grammar is chainable, but a side effect can have only one effect (return, exception, effect) per grammar expression. However, side effects will automatically combine if multiple expressions are applied to the same mock/method.

This enables grammar for building the side effect like:

allow **<Mock>** *to* ...

be_called_with **<Args>** *on_method* **<Name>** *and_return_value* **<Value>**

and_raise_exception(*raised_exception*)
Raise an exception. Alias and_raise.

and_return_value(**return_value*)
Add a return value (or a list of return values to cycle through). Alias and_return.

apply_to(*mock_subject: unittest.mock.Mock*) → unittest.mock.Mock
Apply the built side effect to the given Python Mock.

called_with(**args*, ***kwargs*)
Specify call args that trigger the side effect. Alias when_called_with.

on_method(*method_name*)
Specify method name on the mocked object which will have the side effect applied.

allow(my_mock).to(return_value(5).on_method('foo')) assert my_mock.foo() == 5

with_effect(*effect: Callable*)
Apply a generic effect when invoking the side effect.

4.1.3 allows.side_effect module

class allows.side_effect.**SideEffect**(*call_args=None*, *effect=None*, *default_effect=None*)
Callable object that can compose many side effects with corresponding arguments or default response.

merge(*other: allows.side_effect.SideEffect*) → allows.side_effect.SideEffect
Build a new SideEffect from this one and another

class allows.side_effect.**SideEffectBuilder**(*call_args=None*, *effect=None*)
Uses builder pattern to set up Callable effect and Call argument values for SideEffect objects.

Only one effect can be applied in a given builder (return, exception, effect).

build()
Create the side effect.

with_call_args(**args*, ***kwargs*)
Only invoke side effect when certain args are present

with_effect(*effect: Callable*)
Add a generic effect to the side effect.

with_raised_exception(*raised_exception: Exception*)
Raise an exception.

with_return_value(**return_values*)
Add a return value.

allows.side_effect.**no_op**(**args*, ***kwargs*)

4.1.4 allows.exception module

exception allows.exception.**AllowsException**
Raised when invariants are violated while creating SideEffects (like multiple side effects specified)

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/dvndrsn/allows/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.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dvndrsn/allows/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 :)

5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/allows.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 allows
$ cd allows/
$ 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 allows tests
$ python setup.py test or 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.

5.3 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 2.7, 3.5 and 3.6, 3.7 and for PyPy. Check https://travis-ci.org/dvndrsn/allows/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_allows
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Dave Anderson <dave@dvndrsn.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.0 (2019-06-01)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index