mockextras Documentation

Release 1.0.0

Man AHL

April 28, 2016

Contents

1 DocStrings 3

2 Indices and tables 9

Python Module Index 11

The mockextras library is designed to be used with the unittest.mock library in python 3 or the mock backport of this (http://www.voidspace.org.uk/python/mock/) in python 2. The mockextras library adds a number of features that are found in other mocking libraries namely:

- a fluent API for the configuration of stubs
- stubs
- matchers

CHAPTER 1

DocStrings

mockextras._fluent.**when**(*mock_fn*)

Provides a fluent API for specifying stubs.

For example, you can specify different values to return or exceptions to raise based on the arguments passed into the called_with:

```
>>> try:
...     from unittest.mock Mock
... except ImportError:
...     from mock import Mock
>>>
>>> mock = Mock()
>>> when(mock).called_with("hello").then("world")

>>> when(mock).called_with("foo").then("bar")

>>> when(mock).called_with(100, 200).then(RuntimeError("Boom!"))

>>> mock("hello")
'world'
>>> mock("foo")
'bar'
>>> mock(100, 200)
Traceback (most recent call last):
...
RuntimeError: Boom!
```

You can use 'then' multiple times to specify a sequence of results.

```
>>> mock = Mock()
>>> when(mock).called_with("monkey").then("weezel")\
...                                 .then("badger")\
...                                 .then(RuntimeError("Boom!"))

>>> mock("monkey")
'weezel'
>>> mock("monkey")
'badger'
>>> mock("monkey")
Traceback (most recent call last):
...
RuntimeError: Boom!
```

The last value is repeated for any subsequent calls.

You can use matchers, such as Any(), as wild-card arguments when matching call arguments. The stub's configuration is searched in the order it was specified so you can put more specific call argument specifications ahead of more general ones. For example:

```
>>> from mockextras import Any
>>> mock = Mock()
>>> when(mock).called_with(100, 200).then("monkey")

>>> when(mock).called_with(100, Any()).then("hello")

>>> mock(100, 200)
'monkey'
>>> mock(100, 300)
'hello'
>>> mock(100, "monkey")
'hello'
>>> mock(100, { "key" : 1000 })
'hello'
```

The following matchers are available in mockextras:

- Any
- Contains
- AnyOf

See their documentation for more info.

mockextras._stub.**seq**(*iterable*)

Used to define a sequence of return values for a stub based on an iterable, such as a container:

```
>>> try:
...     from unittest.mock Mock, call
... except ImportError:
...     from mock import Mock, call
>>>
>>> l = range(1, 5)
>>> fn = stub((call(), seq(l)))
>>> fn()
1
>>> fn()
2
>>> fn()
3
```

or a generator:

```
>>> i = xrange(1, 5)
>>> fn = stub((call(), seq(i)))
>>> fn()
1
>>> fn()
2
>>> fn()
3
```

mockextras._stub.**stub**(**args*)

Makes stubs that can be used stand-alone or with mock.

Stubs are dumb functions, used in testing, they do no processing but they can take arguments and return predefined results.

A stub is configured so it returns different values depending on the arguments passed to it. You configure it with one or more pairs of call arguments and results then when the stub is called with a given set of call arguments the corresponding result is returned. If the result is an Exception the result is raised. If more than one result is specified the results will be returned/raised one at a time over successive calls to the stub. If you wish to specify successive results using an iterable you must wrap it with seq().

You can use a stub in place of a function, for example:

```
>>> try:
...     from unittest.mock call
... except ImportError:
...     from mock import call
>>>
>>> fn = stub((call("hello"), "world"),
...           (call("foo"),   1, 2, 4, 8),
...           (call("bar"),   seq(xrange(100))),
...           (call("baz"),   KeyError('baz')),
...           (call("boom"),  100, RuntimeError, 200, ValueError("boom")))
>>> fn("hello")
'world'
>>> fn("foo")
1
>>> fn("foo")
2
>>> fn("foo")
4
```

Or you can combine it with a mock by setting it as the side_effect. This has the advantage that you can later verify the function was called as expected.

```
>>> try:
...     from unittest.mock Mock, call
... except ImportError:
...     from mock import Mock, call
>>>
>>> mock = Mock()
>>> mock.side_effect = stub((call("hello"), "world"),
...                         (call("foo"),   1,2,4,8))
>>> mock("hello")
'world'
>>> mock("foo")
1
>>> mock("foo")
2
>>> mock("foo")
4
>>> assert mock.call_args_list == [call("hello"), call("foo"), call("foo"), call("foo")]
```

Also you can use stubs as methods on Mock objects. Whether you use them directly as the methods or as the side_effect of a mock method depends on whether you want to verify the method calls.

```
>>> mock_obj = Mock(my_first_method=stub((call(50), 100), (call(100), 200)))
>>> mock_obj.my_second_method = stub((call("a"), "aa"), (call("b"), "bb"))
>>> mock_obj.my_third_method.side_effect = stub((call(123), 456), (call(789), 54321))
```

```
>>> mock_obj.my_first_method(50)
100
```

```
>>> mock_obj.my_second_method('b')
'bb'
>>> mock_obj.my_third_method(123)
456
>>> assert mock_obj.mock_calls == [call.my_third_method(123)] # only the mocked call is recorded
```

You can use matchers, such as Any(), as wild-card arguments when matching call arguments. The stub's configuration is searched in the order it was specified so you can put more specific call argument specifications ahead of more general ones.

For example:

```
>>> from mockextras import Any
>>> fn = stub((call(100, 200),   "monkey"),
...           (call(100, Any()), "hello"))
>>> fn(100, 200)
'monkey'
>>> fn(100, 300)
'hello'
>>> fn(100, "monkey")
'hello'
>>> fn(100, { "key" : 1000 })
'hello'
```

The following matchers are available in mockextras:

- Any
- Contains
- AnyOf

See their documentation for more info.

class mockextras._matchers.**Any**(*cls=<type 'object'>*)

Matchers act as wildcards when defining a stub or when asserting call arguments.

The Any matcher will match any object.

```
>>> whatever = Any()
>>> assert whatever == 'hello'
>>> assert whatever ==  100
>>> assert whatever ==  range(10)
```

You can optionally specify a type so that Any only matches objects of that type.

```
>>> anystring = Any(basestring)
>>> assert anystring == 'hello'
>>> assert anystring == 'monkey'
>>> assert anystring == u'bonjour'
>>> assert anystring != ['hello', 'world']
```

Any can be used when specifying stubs:

```
>>> try:
...     from unittest.mock Mock, call
... except ImportError:
...     from mock import Mock, call
>>>
>>> from mockextras import stub
>>> mock = Mock()
```

```
>>> mock.side_effect = stub((call("hello", "world"), 100),
...                         (call("bye bye", Any()), 200))
>>> mock("bye bye", "world")
200
>>> mock("bye bye", "Fred")
200
>>> mock("bye bye", range(100))
200
>>> mock("bye bye", { 'a' : 1000, 'b' : 2000})
200
```

or when asserting call arguments:

```
>>> try:
...     from unittest.mock Mock
... except ImportError:
...     from mock import Mock
>>>
>>> mock = Mock()
>>> mock("bye bye", "world")
<Mock name='mock()' id='...'>
>>> mock.assert_called_once_with("bye bye", Any())
```

```
>>> mock("bye bye", "Fred")
<Mock name='mock()' id='...'>
>>> assert mock.call_args_list == [call("bye bye", "world"),
...                                call("bye bye", Any())]
```

class mockextras._matchers.**Contains**(*value*)

Matchers act as wildcards when defining a stub or when asserting call arguments.

The Contains matcher will match objects that contain the given value or substring.

```
>>> contains_five = Contains(5)
>>> assert contains_five == range(10)
>>> assert contains_five != range(4)
```

```
>>> contains_ello = Contains('ello')
>>> assert contains_ello == "hello"
>>> assert contains_ello != "bye bye"
```

Contains can be used when specifying stubs:

```
>>> try:
...     from unittest.mock Mock, call
... except ImportError:
...     from mock import Mock, call
>>>
>>> from mockextras import stub
>>> mock = Mock()
>>> mock.side_effect = stub((call("hello", "world"), 100),
...                         (call("bye bye", Contains('monkey')), 200))
>>> mock("bye bye", "uncle monkey")
200
```

or when asserting call arguments:

```
>>> try:
...     from unittest.mock Mock
... except ImportError:
```

```
...     from mock import Mock
>>>
>>> mock = Mock()
>>> mock("bye bye", "world")
<Mock name='mock()' id='...'>
>>> mock.assert_called_once_with("bye bye", Contains('or'))
```

```
>>> mock("bye bye", "Fred")
<Mock name='mock()' id='...'>
>>> assert mock.call_args_list == [call("bye bye", "world"),
...                               call("bye bye", Contains('red'))]
```

class mockextras._matchers.**AnyOf**(**args*)

Matchers act as wildcards when defining a stub or when asserting call arguments.

The AnyOf matcher will ....

```
>>> is_a_small_prime = AnyOf(2,3,5,7,11,13)
>>> assert is_a_small_prime == 3
>>> assert is_a_small_prime != 4
```

AnyOf can be used when specifying stubs:

```
>>> try:
...     from unittest.mock Mock, call
... except ImportError:
...     from mock import Mock, call
>>>
>>> from mockextras import stub
>>> mock = Mock()
>>> mock.side_effect = stub((call("hello"), 100),
...                         (call(AnyOf('monkey', 'donkey', 'badger')), 200))
>>> mock("monkey")
200
```

or when asserting call arguments:

```
>>> try:
...     from unittest.mock Mock
... except ImportError:
...     from mock import Mock
>>>
>>> mock = Mock()
>>> mock("donkey")
<Mock name='mock()' id='...'>
>>> mock.assert_called_once_with(AnyOf('monkey', 'donkey', 'badger'))
```

```
>>> mock("monkey")
<Mock name='mock()' id='...'>
>>> assert mock.call_args_list == [call("donkey"),
...                               call(AnyOf('monkey', 'donkey', 'badger'))]
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

m

mockextras._fluent, 3
mockextras._matchers, 6
mockextras._stub, 4

A

Any (class in mockextras._matchers), 6
AnyOf (class in mockextras._matchers), 8

C

Contains (class in mockextras._matchers), 7

M

mockextras._fluent (module), 3
mockextras._matchers (module), 6
mockextras._stub (module), 4

S

seq() (in module mockextras._stub), 4
stub() (in module mockextras._stub), 4

W

when() (in module mockextras._fluent), 3