
Expects Documentation

Release 0.7.2

Jaime Gil de Sagredo

June 26, 2015

1	Usage	3
2	Contents	5
2.1	Installation	5
2.2	Matchers	5
2.3	Aliases	23
2.4	Custom Matchers	24
2.5	3rd Party Matchers	26
2.6	Changes	27
3	Indices and tables	33
	Python Module Index	35

Expects is an *expressive* and *extensible* TDD/BDD assertion library for Python. Expects can be *extended* by defining new matchers.

Usage

Just import the `expect` callable and the built-in matchers and start writing test assertions.

```
from expects import *

expect([]).to(be_empty)

expect(False).not_to(be_true)

expect({
    'name': 'Jack',
    'email': 'jack@example.com'
}).to(have_key('name', match('\w+')))

expect(str).to(have_property('split') & be_callable)

expect(lambda: foo).to(raise_error(NameError))

expect('Foo').to(equal('Bar') | equal('Foo'))
```

Contents

2.1 Installation

2.1.1 PyPI

You can install the last stable release of Expects from PyPI using pip or easy_install:

```
$ pip install expects
```

2.1.2 GitHub

Or install the latest sources from Github:

```
$ pip install -e git+git://github.com/jaimegildesagredo/expects.git#egg=expects
```

Also you can download a source code package from [Github](#) and install it using setuptools:

```
$ tar xvf expects-{version}.tar.gz
$ cd expects
$ python setup.py install
```

2.2 Matchers

2.2.1 equal

```
expect(1).to(equal(1))

expect(1).not_to(equal(2))

expect(1).to(equal(2))
```

Failure

Expected 1 to equal 2

```
expect(1).not_to(equal(1))
```

Failure

Expected 1 not to equal 1

2.2.2 be

```
class Foo(object):
    pass

value = Foo()

expect(value).to(be(value))

expect(1).not_to(be(2))

expect(1).to(be(2))
```

Failure

Expected 1 to be 2

```
expect(value).not_to(be(value))
```

Failure

Expected <Foo object at 0x7ff289cb4310> not to be <Foo object at 0x7ff289cb4310>

2.2.3 be_true

```
expect(True).to(be_true)

expect(False).not_to(be_true)

expect(False).to(be_true)
```

Failure

Expected False to be True

```
expect(True).not_to(be_true)
```

Failure

Expected True not to be True

2.2.4 be_false

```
expect(False).to(be_false)

expect(True).not_to(be_false)
```

```
expect (True) .to (be_false)
```

Failure

Expected True to be False

```
expect (False) .not_to (be_false)
```

Failure

Expected False not to be False

2.2.5 be_none

```
expect (None) .to (be_none)
```

```
expect ('foo') .not_to (be_none)
```

```
expect (True) .to (be_none)
```

Failure

Expected True to be None

```
expect (None) .not_to (be_none)
```

Failure

Expected None not to be None

2.2.6 be_a / be_an

```
class Foo(object):
    pass
```

```
class Bar(object):
    pass
```

```
class Object(object):
    pass
```

```
expect (Foo()) .to (be_a (Foo))
```

```
expect (Foo()) .not_to (be_a (Bar))
```

```
expect (Foo()) .to (be_an (object))
```

```
expect (Foo()) .not_to (be_an (Object))
```

```
expect (Foo()) .to (be_a (Bar))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to be a Bar

```
expect(Foo()).to_not(be_a(Foo))
```

Failure

Expected <Foo object at 0x7ff289cb4310> not to be a Foo

```
expect(Foo()).to(be_an(Object))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to be an Object

```
expect(Foo()).not_to(be_an(object))
```

Failure

Expected <Foo object at 0x7ff289cb4310> not to be an object

2.2.7 be_empty

```
expect('').to(be_empty)

expect(iter('')).to(be_empty)

expect('foo').not_to(be_empty)

expect('foo').to(be_empty)
```

Failure

Expected 'foo' to be empty

```
expect(iter('foo')).to(be_empty)
```

Failure

Expected <str_iterator object at 0x7fd4832d6950> to be empty

```
expect('').to_not(be_empty)
```

Failure

Expected '' not to be empty

2.2.8 be_above

```
expect(5).to(be_above(4))

expect(1).not_to(be_above(4))
```

```
expect(1).to(be_above(4))
```

Failure

Expected 1 to be above 4

```
expect(5).not_to(be_above(4))
```

Failure

Expected 5 not to be above 4

2.2.9 be_below

```
expect(1).to(be_below(4))
```

```
expect(4).not_to(be_below(1))
```

```
expect(4).to(be_below(1))
```

Failure

Expected 4 to be below 1

```
expect(1).not_to(be_below(4))
```

Failure

Expected 1 not to be below 4

2.2.10 be_above_or_equal

```
expect(5).to(be_above_or_equal(4))
```

```
expect(5).to(be_above_or_equal(5))
```

```
expect(1).to_not(be_above_or_equal(4))
```

```
expect(1).to(be_above_or_equal(4))
```

Failure

Expected 1 to be above or equal 4

```
expect(5).not_to(be_above_or_equal(4))
```

Failure

Expected 5 not to be above or equal 4

```
expect(5).not_to(be_above_or_equal(5))
```

Failure

Expected 5 not to be above or equal 5

2.2.11 be_below_or_equal

```
expect(1).to(be_below_or_equal(4))  
expect(5).to(be_below_or_equal(5))  
expect(4).not_to(be_below_or_equal(1))  
expect(4).to(be_below_or_equal(1))
```

Failure

Expected 4 to be below or equal 1

```
expect(1).not_to(be_below_or_equal(4))
```

Failure

Expected 1 not to be below or equal 4

```
expect(5).not_to(be_below_or_equal(5))
```

Failure

Expected 5 not to be below or equal 5

2.2.12 be_within

```
expect(5).to(be_within(4, 7))  
expect(5.5).to(be_within(4, 7))  
expect(1).not_to(be_within(4, 7))  
expect(1).to(be_within(4, 7))
```

Failure

Expected 1 to be within 4, 7

```
expect(5).not_to(be_within(4, 7))
```

Failure

Expected 5 not to be within 4, 7

2.2.13 be_callable

```
expect(lambda: None).to(be_callable)
expect('foo').to(be_callable)
```

Failure

Expected 'foo' to be callable

2.2.14 have_len / have_length

```
expect('foo').to(have_len(3))
expect('foo').to(have_len(be_above_or_equal(3)))
expect(iter('foo')).to(have_length(3))
expect('foo').not_to(have_len(2))
expect('foo').to(have_length(2))
```

Failure

Expected 'foo' to have length 2

```
expect('foo').to(have_len(be_bellow(2)))
```

Failure

Expected 'foo' to have len be bellow 2

```
expect(iter('foo')).to(have_len(2))
```

Failure

Expected <str_iterator object at 0x7fd4832d6950> to have len 2

```
expect('foo').not_to(have_len(3))
```

Failure

Expected 'foo' not to have len 3

2.2.15 have_property

```
class Foo(object):
    def __init__(self, **kwargs):
        for name, value in kwargs.items():
            setattr(self, name, value)

expect(Foo(bar=0, baz=1)).to(have_property('bar'))
```

```
expect(Foo(bar=0, baz=1)).to(have_property('bar', 0))

expect(Foo(bar=0, baz=1)).not_to(have_property('foo'))

expect(Foo(bar=0, baz=1)).not_to(have_property('foo', 0))

expect(Foo(bar=0, baz=1)).not_to(have_property('bar', 1))

expect(Foo(bar=0, baz=1)).to(have_property('bar', be_below(1)))

expect(Foo(bar=0, baz=1)).to(have_property('bar', not_(be_above(1))))

expect(Foo(bar=0, baz=1)).to(have_property('foo'))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have property 'foo'

```
expect(Foo(bar=0, baz=1)).to(have_property('foo', 0))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have property 'foo' equal 0

```
expect(Foo(bar=0, baz=1)).to(have_property('bar', 1))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have property 'bar' equal 1

```
expect(Foo(bar=0, baz=1)).to(have_property('bar', None))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have property 'bar' equal None

```
expect(Foo(bar=0, baz=1)).not_to(have_property('bar'))
```

Failure

Expected <Foo object at 0x7ff289cb4310> not to have property 'bar'

```
expect(Foo(bar=0, baz=1)).not_to(have_property('bar', 0))
```

Failure

Expected <Foo object at 0x7ff289cb4310> not to have property 'bar' equal 0

```
expect(Foo(bar=0, baz=1)).to(have_property('bar', be_above(1)))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have property 'bar' be above 1

```
expect(Foo(bar=0, baz=1)).to(have_property('bar', not_(be_below(1))))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have property 'bar' not be below 1

2.2.16 have_properties

```
class Foo(object):
    def __init__(self, **kwargs):
        for name, value in kwargs.items():
            setattr(self, name, value)

expect(Foo(bar=0, baz=1)).to(have_properties('bar', 'baz'))

expect(Foo(bar=0, baz=1)).to(have_properties(bar=0, baz=1))

expect(Foo(bar=0, baz=1)).to(have_properties('bar', baz=1))

expect(Foo(bar=0, baz=1)).to(have_properties({'bar': 0, 'baz': 1}))

expect(Foo(bar=0, baz=1)).to(have_properties(bar=be_an(int)))

expect(Foo(bar=0, baz=1)).to_not(have_properties('foo', 'foobar'))

expect(Foo(bar=0, baz=1)).to_not(have_properties(foo=0, foobar=1))

expect(Foo(bar=0, baz=1)).not_to(have_properties(foo=0, bar=1))

expect(Foo(bar=0, baz=1)).not_to(have_properties({'foo': 0, 'foobar': 1}))

expect(Foo(bar=0, baz=1)).not_to(have_properties({'foo': 0, 'bar': 1}))

expect(Foo(bar=0, baz=1)).not_to(have_properties('foo', 'bar'))

expect(Foo(bar=0, baz=1)).to(have_properties('bar', 'foo'))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have properties 'bar' and 'foo'

```
expect(Foo(bar=0, baz=1)).to(have_properties(bar=0, foo=1))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have properties 'bar' equal 0 and 'foo' equal 1

```
expect(Foo(bar=0, baz=1)).to(have_properties(bar=1, baz=1))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have properties 'bar' equal 1 and 'baz' equal 1

```
expect(Foo(bar=0, baz=1)).to(have_properties('foo', bar=0))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have properties 'foo' and 'bar' equal 0

```
expect(Foo(bar=0, baz=1)).to(have_properties('baz', bar=1))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have properties 'baz' and 'bar' equal 1

```
expect(Foo(bar=0, baz=1)).to(have_properties({'bar': 1, 'baz': 1}))
```

Failure

Expected <Foo object at 0x7ff289cb4310> to have properties 'bar' equal 1 and 'baz' equal 1

```
expect(Foo(bar=0, baz=1)).not_to(have_properties('bar', 'baz'))
```

Failure

Expected <Foo object at 0x7ff289cb4310> not to have properties 'bar' and 'baz'

```
expect(Foo(bar=0, baz=1)).not_to(have_properties(bar=0, baz=1))
```

Failure

Expected <Foo object at 0x7ff289cb4310> not to have properties 'bar' equal 0 and 'baz' equal 1

```
expect(Foo(bar=0, baz=1)).to(have_properties(bar=be_a(str)))
```

Failure

Expected <Foo object at 0x7ff289cb4310> not to have properties 'bar' be a str

2.2.17 have_key

```
expect({'bar': 0, 'baz': 1}).to(have_key('bar'))
expect({'bar': 0, 'baz': 1}).to(have_key('bar', 0))
expect({'bar': 0, 'baz': 1}).not_to(have_key('foo'))
expect({'bar': 0, 'baz': 1}).not_to(have_key('foo', 0))
expect({'bar': 0, 'baz': 1}).to_not(have_key('bar', 1))
expect({'bar': 0, 'baz': 1}).to(have_key('bar', be_below(1)))
expect({'bar': 0, 'baz': 1}).to(have_key('foo'))
```

Failure

Expected {'bar': 0, 'baz': 1} to have key 'foo'

```
expect({'bar': 0, 'baz': 1}).to(have_key('foo', 0))
```

Failure

Expected {'bar': 0, 'baz': 1} to have key 'foo' equal 0

```
expect({'bar': 0, 'baz': 1}).to(have_key('bar', 1))
```

Failure

Expected {'bar': 0, 'baz': 1} to have key 'bar' equal 1

```
expect({'bar': 0, 'baz': 1}).to(have_key('bar', None))
```

Failure

Expected {'bar': 0, 'baz': 1} to have key 'bar' equal None

```
expect('My foo string').to(have_key('foo', 0))
```

Failure

Expected {'bar': 0, 'baz': 1} to have key 'foo' equal 0 but is not a dict

```
expect({'bar': 0, 'baz': 1}).not_to(have_key('bar'))
```

Failure

Expected {'bar': 0, 'baz': 1} not to have key 'bar'

```
expect({'bar': 0, 'baz': 1}).not_to(have_key('bar', 0))
```

Failure

Expected {'bar': 0, 'baz': 1} not to have key 'bar' equal 0

```
expect('My foo string').not_to(have_key('foo', 0))
```

Failure

Expected {'bar': 0, 'baz': 1} not to have key 'foo' equal 0 but is not a dict

```
expect({'bar': 0, 'baz': 1}).to(have_key('bar', be_above(1)))
```

Failure

Expected {'bar': 0, 'baz': 1} to have key 'bar' be above 1

2.2.18 have_keys

```
expect({'bar': 0, 'baz': 1}).to(have_keys('bar', 'baz'))
expect({'bar': 0, 'baz': 1}).to(have_keys(bar=0, baz=1))
expect({'bar': 0, 'baz': 1}).to(have_keys('bar', baz=1))
expect({'bar': 0, 'baz': 1}).to(have_keys({'bar': 0, 'baz': 1}))
expect({'bar': 0, 'baz': 1}).not_to(have_keys('foo', 'foobar'))
expect({'bar': 0, 'baz': 1}).not_to(have_keys(foo=0, foobar=1))
expect({'bar': 0, 'baz': 1}).not_to(have_keys(foo=0, bar=1))
expect({'bar': 0, 'baz': 1}).not_to(have_keys({'foo': 0, 'foobar': 1}))
expect({'bar': 0, 'baz': 1}).not_to(have_keys('foo', 'bar'))
expect({'bar': 0, 'baz': 1}).to(have_keys('bar', 'foo'))
```

Failure

Expected {'bar': 0, 'baz': 1} to have keys 'bar' and 'foo'

```
expect({'bar': 0, 'baz': 1}).to(have_keys(bar=0, foo=1))
```

Failure

Expected {'bar': 0, 'baz': 1} to have keys 'bar' equal 0 and 'foo' equal 1

```
expect({'bar': 0, 'baz': 1}).to(have_keys(bar=1, baz=1))
```

Failure

Expected {'bar': 0, 'baz': 1} to have keys 'bar' equal 1 and 'baz' equal 1

```
expect({'bar': 0, 'baz': 1}).to(have_keys('foo', 'fuu', bar=0))
```

Failure

Expected {'bar': 0, 'baz': 1} to have keys 'foo', 'fuu' and 'bar' equal 0

```
expect({'bar': 0, 'baz': 1}).to(have_keys('baz', bar=1))
```

Failure

Expected {'bar': 0, 'baz': 1} to have keys 'baz' and 'bar' equal 1

```
expect({'bar': 0, 'baz': 1}).to(have_keys({'bar': 1, 'baz': 1}))
```

Failure

Expected {'bar': 0, 'baz': 1} to have keys 'bar' equal 1 and 'baz' equal 1

```
expect('My foo string').to(have_keys({'bar': 1, 'baz': 1}))
```

Failure

Expected {'bar': 0, 'baz': 1} to have keys 'bar' equal 1 and 'baz' equal 1 but is not a dict

```
expect({'bar': 0, 'baz': 1}).not_to(have_keys('bar', 'baz'))
```

Failure

Expected {'bar': 0, 'baz': 1} not to have keys 'bar' and 'baz'

```
expect({'bar': 0, 'baz': 1}).not_to(have_keys(bar=0, baz=1))
```

Failure

Expected {'bar': 0, 'baz': 1} not to have keys 'bar' equal 0 and 'baz' equal 1

```
expect('My foo string').not_to(have_keys({'bar': 1, 'baz': 1}))
```

Failure

Expected {'bar': 0, 'baz': 1} not to have keys 'bar' equal 1 and 'baz' equal 1 but is not a dict

2.2.19 contain

```
expect(['bar', 'baz']).to(contain('bar'))
expect(['bar', 'baz']).to(contain('bar', 'baz'))
expect(['bar', 'baz']).to(contain('baz', 'bar'))
expect([{'foo': 1}, 'bar']).to(contain({'foo': 1}))
expect(iter(['bar', 'baz'])).to(contain('bar'))
expect(iter(['bar', 'baz'])).to(contain('bar', 'baz'))
expect('My foo string').to(contain('foo'))
expect('My foo string').to(contain('foo', 'string'))
expect(['bar', 'baz']).not_to(contain('foo'))
expect(['bar', 'baz']).not_to(contain('foo', 'foobar'))
expect(['bar', 'baz']).not_to(contain('bar', 'foo'))
expect(['bar', 'baz']).to(contain(be_a(str)))
expect(['bar', 'baz']).to(contain('bar', 'foo'))
```

Failure

Expected ['bar', 'baz'] to contain 'bar' and 'foo'

```
expect(iter(['bar', 'baz'])).to(contain('bar', 'foo'))
```

Failure

Expected ['bar', 'baz'] to contain 'bar' and 'foo'

```
expect(object()).to(contain('bar'))
```

Failure

Expected <object object at 0x7f5004aa1070> to contain 'bar' but is not a valid sequence type

```
expect(['bar', 'baz']).not_to(contain('bar'))
```

Failure

Expected ['bar', 'baz'] not to contain 'bar'

```
expect(['bar', 'baz']).not_to(contain('bar', 'baz'))
```

Failure

Expected ['bar', 'baz'] not to contain 'bar' and 'baz'

```
expect(object()).not_to(contain('bar'))
```

Failure

Expected <object object at 0x7f5004aa1070> not to contain 'bar' but is not a valid sequence type

```
expect(['bar', 'baz']).to(contain(be_an(int), have_len(5)))
```

Failure

Expected ['bar', 'baz'] to contain be an int and have len 5

2.2.20 contain_exactly

```
expect(['bar']).to(contain_exactly('bar'))

expect(['bar', 'baz']).to(contain_exactly('bar', 'baz'))

expect('My foo string').to(contain_exactly('My foo string'))

expect('My foo string').to(contain_exactly('My foo', ' string'))

expect(['bar', 'baz']).to(contain_exactly(equal('bar'), equal('baz')))

expect(['bar', 'baz']).to(contain_exactly('foo'))
```

Failure

Expected ['bar', 'baz'] to contain exactly 'foo'

```
expect(['bar', 'baz']).to(contain_exactly('foo', 'fuu'))
```

Failure

Expected ['bar', 'baz'] to contain exactly 'foo' and 'fuu'

```
expect(['bar', 'baz']).to(contain_exactly('baz', 'bar'))
```

Failure

Expected ['bar', 'baz'] to contain exactly 'baz' and 'bar'

```
expect(['bar', 'baz']).to(contain_exactly('bar'))
```

Failure

Expected ['bar', 'baz'] to contain exactly 'bar'

```
expect(['bar', 'baz', 'foo']).to(contain_exactly('bar', 'baz'))
```

Failure

Expected ['bar', 'baz', 'foo'] to contain exactly 'bar' and 'baz'

```
expect(['bar', 'baz', 'foo', 'fuu']).to(contain_exactly('bar', 'baz', 'foo'))
```

Failure

Expected ['bar', 'baz', 'foo', 'fuu'] to contain exactly 'bar', 'baz' and 'foo'

```
expect('My foo string').to(contain_exactly('foo'))
```

Failure

Expected 'My foo string' to contain exactly 'foo'

```
expect(object()).to(contain_exactly('bar'))
```

Failure

Expected <object object at 0x7f5004aa1070> to contain exactly 'bar' but is not a valid sequence type

```
expect(['bar', 'baz']).to(contain_exactly(equal('baz'), equal('baz')))
```

Failure

Expected ['bar', 'baz'] to contain exactly equal 'bar' and equal 'baz'

2.2.21 contain_only

```
expect(['bar']).to(contain_only('bar'))

expect(['bar', 'baz']).to(contain_only(['baz', 'bar']))

expect(iter(['bar', 'baz'])).to(contain_only('bar', 'baz'))

expect('My foo string').to(contain_only('My foo string'))

expect('My foo string').to(contain_only('My foo', ' string'))

expect(['bar', 'baz']).to(contain_only(equal('bar'), equal('baz')))

expect(['bar', 'baz']).to(contain_only('foo'))
```

Failure

Expected ['bar', 'baz'] to contain only 'foo'

```
expect(['bar', 'baz', 'foo']).to(contain_only('bar', 'baz'))
```

Failure

Expected ['bar', 'baz', 'foo'] to contain only 'bar' and 'baz'

```
expect('My foo string').to(contain_only('foo'))
```

Failure

Expected 'My foo string' to contain only 'foo'

```
expect(object()).to(contain_only('bar'))
```

Failure

Expected <object object at 0x7f5004aa1070> to contain only 'bar' but is not a valid sequence type

```
expect(['bar', 'baz']).to(contain_only(equal('baz'), equal('foo')))
```

Failure

Expected ['bar', 'baz'] to contain only equal 'baz' and equal 'foo'

2.2.22 start_with

```
expect('My foo string').to(start_with('My foo'))

expect('My foo string').not_to(start_with('tring'))

expect([1, 2, 3]).to(start_with(1))

expect([1, 2, 3]).to(start_with(1, 2))
```



```
expect(OrderedDict([('bar', 0), ('baz', 1)]).to(start_with('bar', 'baz'))

expect(iter([1, 2, 3])).to(start_with(1, 2))

expect([1, 2, 3]).not_to(start_with(2, 3))

expect([1, 2, 3]).not_to(start_with(1, 1))

expect('My foo string').to(start_with('tring'))
```

Failure

Expected 'My foo string' to start with 'tring'

```
expect([1, 2, 3]).to(start_with(2))
```

Failure

Expected [1, 2, 3] to start with 2

```
expect([1, 2, 3]).to(start_with(2, 3))
```

Failure

Expected [1, 2, 3] to start with 2 and 3

```
expect([1, 2, 3]).to(start_with(1, 1))
```

Failure

Expected [1, 2, 3] to start with 1 and 1

```
expect({'bar': 0, 'baz': 1}).to(start_with('bar', 'baz'))
```

Failure

Expected {'bar': 0, 'baz': 1} to start with 'bar' and 'baz' but it does not have ordered keys

2.2.23 end_with

```
expect('My foo string').to(end_with('tring'))

expect('My foo string').not_to(end_with('My foo'))

expect([1, 2, 3]).to(end_with(3))

expect([1, 2, 3]).to(end_with(2, 3))

expect(OrderedDict([('bar', 0), ('baz', 1)]).to(end_with('bar', 'baz'))

expect([1, 2, 3]).to_not(end_with(1, 2))

expect([1, 2, 3]).to_not(end_with(3, 3))
```

```
expect('My foo string').to(end_with('My fo'))
```

Failure

Expected 'My foo string' to end with 'My fo'

```
expect([1, 2, 3]).to(end_with(3, 3))
```

Failure

Expected [1, 2, 3] to end with 3 and 3

```
expect({'bar': 0, 'baz': 1}).to(end_with('baz', 'bar'))
```

Failure

Expected {'bar': 0, 'baz': 1} to end with 'baz' and 'bar' but it does not have ordered keys

2.2.24 match

```
expect('My foo string').to(match(r'My \w+ string'))
expect('My foo string').to(match(r'\w+ string'))
expect('My foo string').to(match(r'my [A-Z]+ string', re.I))
expect('My foo string').not_to(match(r'My \W+ string'))
expect('My foo string').not_to(match(r'My \W+ string', re.I))
expect('My foo string').to(match(pattern))
```

Failure

Expected 'My foo string' to match r'My \W+ string'

```
expect('My foo string').not_to(match(r'My \w+ string'))
```

Failure

Expected 'My foo string' not to match r'My \w+ string'

2.2.25 raise_error

```
def callback():
    raise AttributeError('error message')

expect(callback).to(raise_error)

expect(callback).to(raise_error(AttributeError))

expect(callback).to(raise_error(AttributeError, 'error message'))
```

```
expect(callback).to(raise_error(AttributeError, match(r'error \w+')))
```

```
def callback():
    raise AttributeError(2)
```

```
expect(callback).to(raise_error(AttributeError, 2))
```

```
def callback():
    raise KeyError()
```

```
expect(callback).to(raise_error(AttributeError))
```

Failure

Expected <function callback at 0x7fe70cb103b0> to raise AttributeError but KeyError raised

```
expect(lambda: None).to(raise_error(AttributeError))
```

Failure

Expected <function <lambda> at 0x7f3e670863b0> to raise AttributeError but not raised

```
def callback():
    raise AttributeError('bar')
```

```
expect(callback).to(raise_error(AttributeError, 'foo'))
```

Failure

Expected <function callback at 0x7fe70cb103b0> to raise AttributeError with message 'foo' but message was 'bar'

2.3 Aliases

The `aliases` module contains a set of matcher *aliases* that are commonly used when composing matchers and are not meant to be imported every time.

To use the aliases just import them:

```
from expects import *
from expects.aliases import *
```

```
expect([1, 2]).to(contain_exactly(an(int), 2))
```

The same code without using the `an` alias for the `be_an` matcher:

```
from expects import *
```

```
expect([1, 2]).to(contain_exactly(be_an(int), 2))
```

2.3.1 Reference

class `expects.aliases.a(expected)`

```
class expects.aliases.an(expected)
class expects.aliases.above(expected)
class expects.aliases.above_or_equal(expected)
class expects.aliases.below(expected)
class expects.aliases.below_or_equal(expected)
```

2.4 Custom Matchers

2.4.1 Introduction

Expects can be *extended* by defining *new matchers*. The `matchers` module contains the bases for building custom matchers.

2.4.2 Tutorial

The easiest way to define a new matcher is to extend the *Matcher* class and override the *Matcher.__match()* method.

For example, to define a matcher to check if a *request* object contains a given header takes <10 lines of code:

```
from expects.matchers import Matcher

class have_header(Matcher):
    def __init__(self, expected):
        self._expected = expected

    def _match(self, request):
        return self._expected in request.headers
```

And then you only need to import the new defined matcher and write your expectation:

```
from expects import expect
from my_custom_matchers import have_header

expect(my_request).to(have_header('Content-Type'))
```

2.4.3 Advanced

For more complex matchers you can override the *Matcher* methods in order to achieve the needed behavior.

class `expects.matchers.Matcher`

The *Matcher* class is the base class for all *Expects* matchers.

It defines a set of methods to ease writing new matchers.

`__description` (*subject*)

This method receives the *subject* of the expectation and returns a string with the description of the matcher to be used in failure messages.

By default returns a string with the following format:

```
'{name} {expected}'
```

Where *name* is based on the matcher class name and *expected* is the value passed to the constructor.

Parameters *subject* – The target value of the expectation.

Return type a string

`_failure_message` (*subject*)

This method will be called from an expectation *only* when the expectation is going to fail. It should return a string with the failure message.

By default returns a failure message with the following format:

```
'Expected {subject} to {description}'
```

With the passed *subject* and the result of calling the `_description()` method.

Parameters *subject* – The target value of the expectation.

Return type a string

`_failure_message_negated` (*subject*)

Like the `_failure_message()` method but will be called when a negated expectation is going to fail. It should return a string with the failure message for the negated expectation.

By default returns a failure message with the following format:

```
'Expected {subject} not to {description}'
```

Parameters *subject* – The target value of the expectation.

Return type a string

`_match` (*subject*)

This method will be called when the matcher is used in an expectation. It should be overwritten to implement the matcher logic. If not raises `NotImplementedError`.

Receives the expectation *subject* as the unique positional argument and should return `True` if the matcher matches the subject and `False` if it does not.

Parameters *subject* – The target value of the expectation.

Return type a boolean

`_match_negated` (*subject*)

Like `_match()` but will be called when used in a negated expectation. It can be used to implement a custom logic for negated expectations.

By default returns the result of `not self._match(subject)`.

Parameters *subject* – The target value of the expectation.

Return type a boolean

`_match_value` (*matcher*, *value*)

This method receives a *Matcher* instance and a *value to be matched* as first and second arguments respectively, and returns `True` or `False` depending on whether the *value* matches.

If the argument passed as *matcher* does not implements the *Matcher* interface then the `equal` built-in matcher is used.

Examples:

```
>>> self._match_value('foo', 'foo')
True
>>> self._match_value('foo', 'bar')
False
>>> self._match_value(match('\w+'), 'foo')
True
```

Parameters

- **matcher** – A matcher that will be used to match the given value.
- **value** – A value to test if matches.

Return type `bool`

2.4.4 Testing

The `testing` module provides helpers to ease the testing of your custom matchers.

class `expects.testing.failure(message)`

The `failure` context manager can be used to build assertions of your expectation failures. It tests that the code inside the context manager raises an `AssertionError` and matches the given message (whether any has been specified).

Parameters `message` (an `expects.matchers.Matcher` or string) – should match the failure message. If a string is passed, the `end_with` matcher will be used by default.

Raises `AssertionError` when no `AssertionError` was raised, the failure message didn't match or another exception raised.

Note: The `failure` context manager can be used without being *called* (for example, if you don't want to specify a *failure message*).

Examples:

```
>>> with failure:
...     expect(object()).to(have_property('foo'))
```

```
>>> with failure("to have property 'foo'"):
...     expect(object()).to(have_property('foo'))
```

```
>>> with failure(end_with("have property 'foo'")):
...     expect(object()).to(have_property('foo'))
```

```
>>> with failure("to have property '__class__'):
...     expect(object()).to(have_property('__class__'))
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
  File "expects/testing.py", line 40, in __exit__
    raise AssertionError('Expected AssertionError to be raised')
AssertionError: Expected AssertionError to be raised
```

2.5 3rd Party Matchers

Tornado Expects Matchers for Tornado request and response objects.

Doublex Expects Matchers for Doublex test doubles assertions.

Server Expects Serverspec-like Expects matchers library.

2.6 Changes

2.6.1 0.7.2 (Jun 26, 2015)

Bug fixes

- *GH-26* <<https://github.com/jaimegildesagredo/expectjs/issues/26>>_

Bug fixes

- The `contain_exactly` matcher does not raise an `IndexError` if the subject list has fewer elements than the expected one. [GH-23](#).

2.6.2 0.7.1 (Jun 9, 2015)

Bug fixes

- The `contain_exactly` matcher does not raise an `IndexError` if the subject list has fewer elements than the expected one. [GH-23](#).

2.6.3 0.7.0 (Mar 1, 2015)

Highlights

- Added `have_len` as an alias to `have_length`.
- The `have_len` and `have_length` matchers can receive another matcher as expected value:

```
expect('foo').to(have_len(be_above(2)))
```

- The `contain` and `contain_exactly` matchers now can receive another matchers as arguments:

```
expect(['foo', 'bar']).to(contain(be_a(str)))
expect(['foo', 'bar']).to(contain_exactly(be_a(str), match('\w+')))
```

- Improved `be_a` and `be_an` failure messages.
- Added the `contain_only` matcher:

```
expect([1, 2]).to(contain_only(2, 1))
```

- Added the `to_not` alias for `not_to` to negate assertions:

```
expect(True).to_not(be_false)
```

- Added the `aliases` module with matcher aliases useful to compose matchers:

```
from expects import *
from expects.aliases import *

expect([1, 2]).to(contain_exactly(an(int), 2))
```

Backwards-incompatible

- The failure context manager now uses the `end_with` matcher as default matcher for failure message instead of the previously used `contain` matcher. Example:

```
>>> from expects.testing import failure
>>> with failure('foo'):
...     raise AssertionError('A foo message')
AssertionError: Expected message 'A foo message' to end with 'foo'

>>> with failure('message'):
...     raise AssertionError('A foo message')
```

2.6.4 0.6.2 (Dec 10, 2014)

Bug fixes

- Fixed `contain_exactly` to work with iterable objects. Regression introduced in v0.6.1.

2.6.5 0.6.1 (Nov 30, 2014)

Bug fixes

- Now the `contain` and `contain_exactly` matchers fail with a proper message when used with a non-sequence type. See [GH-21](#).

2.6.6 0.6.0 (Nov 24, 2014)

Highlights

- Now the `raise_error` matcher can be used without specifying an exception class for writing less strict assertions:

```
expect(lambda: foo).to(raise_error)
```

- Implemented the `Matcher._match_value` method to help develop custom matchers that receive another matchers. See the [docs](#) for more info.
- The `specs` and `docs` directories are now distributed with the source tarball. See [GH-20](#).

2.6.7 0.5.0 (Sep 20, 2014)

Highlights

- Now the `&` and `|` operators can be used to write simpler assertions:


```
expect('Foo').to(have_length(3) & start_with('F'))
expect('Foo').to(equal('Foo') | equal('Bar'))
```

- The `testing.failure` context manager can be used even without calling it with the failure message as argument:

```
with failure:
    expect('foo').to(be_empty)
```

- Also can receive matchers as argument:

```
with failure(end_with('empty')):
    expect('foo').to(be_empty)
```

Note: See also backwards-incompatible changes for `testing.failure`.

- Added the `be_callable` matcher.
- Published a list of [3rd Party Matchers libraries](#).

Bug fixes

- The `be_within` matcher now supports float values.
- In some places `bytes` were not being treated as a string type in python 3.

Backwards-incompatible

- The `match` matcher now passes if matches a part of the subject string instead of all of it. Previously used the `re.match()` and now uses `re.search()`. If your tests depended on this you can migrate them by adding a `'^'` and `'$'` at the beginning and end of your regular expression.
- The `testing.failure` context manager not longer tries to match regular expressions. Instead you can pass the `match` matcher with your regexp.

2.6.8 0.4.2 (Ago 16, 2014)

Highlights

- Added the `not_` matcher to negate another matcher when composing matchers.

2.6.9 0.4.1 (Ago 16, 2014)

Bug fixes

- Now from `expects import *` only imports the `expect` callable and *built in* matchers.

2.6.10 0.4.0 (Ago 15, 2014)

Warnings

This release *does not* maintain backwards compatibility with the previous version because a *new syntax was implemented* based on matchers. Matchers have been implemented maintaining compatibility with its equivalent assertions (and those that break compatibility are listed below). For most users upgrade to this version will only involve a migration to the new syntax.

Highlights

- Improved failure message for `have_keys` and `have_properties` matchers.
- The `raise_error` matcher now can receive any other matcher as the second argument.

Bug fixes

- The `have_key` and `have_keys` always fail if the subject is not a dict.
- Fixed `contain` matcher behavior when negated. See [this commit](#).

Backwards-incompatible

- The `end_with` matcher should receive args in the right order and not reversed. See [this commit](#).
- The `to.have` and `to.have.only` assertions have been renamed to `contain` and `contain_exactly` matchers.
- Assertion chaining has been replaced by *matcher composition* in all places where was possible in the previous version.
- The `testing.failure` context manager now only receives a string matching the failure message.

2.6.11 0.3.0 (Jun 29, 2014)

Highlights

- The `start_with` and `end_with` assertions now support lists, iterators and ordered dicts. [GH-16](#).

Bug fixes

- Fixes a regression in the `raise_error` assertion introduced in v0.2.2 which caused some tests to fail. See [GH-17](#) for more info.

2.6.12 0.2.3 (Jun 4, 2014)

Highlights

- Added the `start_with` and `end_with` assertions. [GH-14](#).

2.6.13 0.2.2 (May 20, 2014)

Bug fixes

- *to.raise_error* now works with a non-string object as second arg. See docs for [examples](#).

2.6.14 0.2.1 (Mar 22, 2014)

Highlights

- Added a *testing* module with the *failure* contextmanager.
- Added a *matchers* module and the *key* matcher.

Bug fixes

- *to.have* and *to.only.have* now work properly when actual is a string.

2.6.15 0.2.0 (Feb 5, 2014)

Highlights

- Added initial plugins support. See [plugins docs](#) for more info.
- The *key* and *property* expectations now return a new *Expects* object that can be used to chain expectations.
- Now every expectation part can be prefixed with *not_* in order to negate an expectation. Ex: `expect('foo').not_to.be.empty` is the same than `expect('foo').to.not_be.empty`.
- Added the *only.have* expectation to test that the subject *only* has the given items.

Backwards-incompatible

- The *greater_than*, *greater_or_equal_to*, *less_than* and *less_or_equal_to* expectations are renamed to *above*, *above_or_equal*, *below* and *below_or_equal*.

2.6.16 0.1.1 (Ago 20, 2013)

Bug fixes

- *to.have* when iterable items are not hashable ([Issue #8](#)).
- *to.have.key* weird behavior when actual is not a *dict* ([Issue #10](#)).

2.6.17 0.1.0 (Ago 11, 2013)

Highlights

- First *expects* release.

Indices and tables

- `genindex`
- `modindex`
- `search`

e

`expects.aliases`, [23](#)
`expects.matchers`, [24](#)
`expects.testing`, [26](#)

Symbols

`_description()` (expects.matchers.Matcher method), [24](#)
`_failure_message()` (expects.matchers.Matcher method),
[25](#)
`_failure_message_negated()` (expects.matchers.Matcher
method), [25](#)
`_match()` (expects.matchers.Matcher method), [25](#)
`_match_negated()` (expects.matchers.Matcher method),
[25](#)
`_match_value()` (expects.matchers.Matcher method), [25](#)

A

`a` (class in expects.aliases), [23](#)
`above` (class in expects.aliases), [24](#)
`above_or_equal` (class in expects.aliases), [24](#)
`an` (class in expects.aliases), [23](#)

B

`below` (class in expects.aliases), [24](#)
`below_or_equal` (class in expects.aliases), [24](#)

E

`expects.aliases` (module), [23](#)
`expects.matchers` (module), [24](#)
`expects.testing` (module), [26](#)

F

`failure` (class in expects.testing), [26](#)

M

`Matcher` (class in expects.matchers), [24](#)