Expects Documentation

Release 0.9.0

Jaime Gil de Sagredo

Oct 25, 2018

Contents

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

CHAPTER 1

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'))
```

CHAPTER 2

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 donwload 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))
```

(continues on next page)

(continued from previous page)

```
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)
```

(continues on next page)

(continued from previous page)

```
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))
```

(continues on next page)

(continued from previous page)

```
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))
```

(continues on next page)

(continued from previous page)

```
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))
```

(continues on next page)

(continued from previous page)

```
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'))
```

(continues on next page)

(continued from previous page)

```
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):
        if self._expected in request.headers:
            return True, ['header found']
        return True, ['header not found']
```

An 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 writting new matchers.

_failure_message(*subject*, *reasons*)

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}
     but: {reasons}
```

With the passed *subject* repr, this matcher repr as *description* and the passed *reasons* from the matcher result.

Parameters

- **subject** (*a string*) – The target value of the expectation.

- **reasons** (*list of strings*) – A list of reasons that caused this matcher to fail.

Return type a string

_failure_message_negated(*subject*, *reasons*)

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} to {description}
     but: {reasons}
```

Parameters

- **subject** (*a string*) – The target value of the expectation.
- **reasons** (*list of strings*) – A list of reasons that caused this matcher to fail.

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 a `tuple` with the `bool` result of the matcher and a `list` of reasons for this result.

If the matcher matches the subject then the boolean result should be `True`. The reasons should be a list with 0 or more strings.

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

Return type tuple (bool, [str])

_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 negating `self._match(subject)`.

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

Return type tuple (bool, [str])

class expects.matchers.**_And**(*op1*, *op2*)

_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 a `tuple` with the `bool` result of the matcher and a `list` of reasons for this result.

If the matcher matches the subject then the boolean result should be `True`. The reasons should be a list with 0 or more strings.

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

Return type tuple (bool, [str])

class expects.matchers.**_Or**(*op1*, *op2*)

_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 a `tuple` with the `bool` result of the matcher and a `list` of reasons for this result.

If the matcher matches the subject then the boolean result should be `True`. The reasons should be a list with 0 or more strings.

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

Return type tuple (bool, [str])

2.4.4 Testing

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

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 0.9.0 (2018-10-25)

Highlights

- Fix Python 3.7 collections ABC classes deprecation warning. See *GH-55 <https://github.com/jaimegildesagredo/expects/pull/55>*.
- Hide `expects` internals from pytest tracebacks. See *GH-51 <https://github.com/jaimegildesagredo/expects/pull/51>*.

Backwards-incompatible

- Dropped Python 2.6 support.

2.6.2 0.8.0 (2016-05-15)

Changes since last stable release:

Highlights

- New failure messages for matchers. Now its easier to see the reason that caused the assertion failure. For example:

```
>>> expect([1, {'foo': 1}]).to(contain(have_key('foo', 2)))

AssertionError:
expected: [1, {'foo': 1}] to contain have key 'foo' equal 2
     but: item have key 'foo' equal 2 not found
```

Bug fixes

- Now the failure message for `have_key`/`have_keys` is fixed when composed with another matcher. See GH-29.
- Allow `contain`, `contain_exactly` and `contain_only` matchers to work with dict views (e.g. `dict.keys()`). See GH-42.
- Allow `contain`, `contain_exactly` and `contain_only` matchers to work with sets. See GH-38.
- Show traceback in `raise_error` when another exception is raised. See GH-41.
- The `equal` matcher now uses `__ne__` for negated assertions. See GH-40.
- The `not_` matcher now uses the inner matcher `_match_negated` method to perform a negated assertion.
- Python 2.6 support.

Backwards-incompatible

Although your assertions should still be working (if are broken, just report an issue), the *custom matchers* api has been changed. To see an example of how to migrate your custom matchers to the new api you can see the doublex-expects migration.

- The `Matcher._match` method now should return a tuple of `bool` representing the result of the matcher and a `list` of reasons that explain this result:

```
def _match(self, subject):
    if subject:
        return True, ['a reason']
    return False, ['another reason']
```

- The `Matcher._description` method was removed. Now, with the change announced above, a matcher description won't need the subject to describe itself, so the `__repr__` magic method will be used instead to describe matchers.
- The `Matcher._match_value` method was removed. With the new api it made much less sense so it was removed and the `expects.matchers.default_matcher` wrapper function was added:

```
>>> default_matcher(1)._match(2)
False, ['was 1']
```

2.6.3 0.8.0rc5 (2016-05-07)

Bug fixes

- Allow `contain`, `contain_exactly` and `contain_only` matchers to work with dict views (e.g. `dict.keys()`). See GH-42.
- Allow `contain`, `contain_exactly` and `contain_only` matchers to work with sets. See GH-38.

- Show traceback in `raise_error` when another exception is raised. See GH-41.

2.6.4 0.8.0rc4 (2015-10-14)

Bug fixes

- Show the correct failure message on negated `contain_exactly` and `contain_only` matchers. See GH-33.

2.6.5 0.8.0rc3 (2015-10-07)

Bug fixes

- The `equal` matcher now uses `__ne__` for negated assertions. See GH-40.
- The `not_` matcher now uses the inner matcher `_match_negated` method to perform a negated assertion.

2.6.6 0.8.0rc2 (2015-08-14)

Bug fixes

- Python 2.6 support.

2.6.7 0.8.0rc1 (2015-07-17)

Highlights

- New failure messages for matchers. Now its easier to see the reason that caused the assertion failure. For example:

```
>>> expect([1, {'foo': 1}]).to(contain(have_key('foo', 2)))

AssertionError:
expected: [1, {'foo': 1}] to contain have key 'foo' equal 2
     but: item have key 'foo' equal 2 not found
```

Bug fixes

- Now the failure message for `have_key`/`have_keys` is fixed when composed with another matcher. See GH-29.

Backwards-incompatible

Although your assertions should still be working (if are broken, just report an issue), the *custom matchers* api has been changed. To see an example of how to migrate your custom matchers to the new api you can see the doublex-expects migration.

- The `Matcher._match` method now should return a tuple of `bool` representing the result of the matcher and a `list` of reasons that explain this result:

```
def _match(self, subject):
    if subject:
        return True, ['a reason']
    return False, ['another reason']
```

- The `Matcher._description` method was removed. Now, with the change announced above, a matcher description won't need the subject to describe itself, so the `__repr__` magic method will be used instead to describe matchers.
- The `Matcher._match_value` method was removed. With the new api it made much less sense so it was removed and the `expects.matchers.default_matcher` wrapper function was added:

```
>>> default_matcher(1)._match(2)
False, ['was 1']
```

2.6.8 0.7.0 (2015-06-26)

Bug fixes

- GH-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.

2.6.9 0.7.1 (2015-06-09)

Bug fixes

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

2.6.10 0.7.0 (2015-03-01)

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')
```

2.6.11 0.6.2 (2014-12-10)

Bug fixes

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

2.6.12 0.6.1 (2014-11-30)

Bug fixes

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

2.6.13 0.6.0 (2014-11-24)

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.

2.6.14 0.5.0 (2014-09-20)

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.

2.6.15 0.4.2 (2014-08-16)

Highlights

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

2.6.16 0.4.1 (2014-08-16)

Bug fixes

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

2.6.17 0.4.0 (2014-08-15)

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 remamed 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.

2.6.18 0.3.0 (2014-06-29)

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.

2.6.19 0.2.3 (2014-06-04)

Highlights

- Added the start_with and end_with assertions. GH-14.

2.6.20 0.2.2 (2014-05-20)

Bug fixes

- *to.raise_error* now works with a non-string object as second arg. See docs for examples.

2.6.21 0.2.1 (2014-03-22)

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.

2.6.22 0.2.0 (2014-02-05)

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`.

2.6.23 0.1.1 (2013-08-20)

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).

2.6.24 0.1.0 (2013-08-11)

Highlights

- First *expects* release.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

Symbols

A

B

E

M