Expecter Documentation

Release 0.2.2

Gary Bernhardt

June 24, 2014

Contents

Release v0.2.2

expecter is a library to help you write assertions. Never again will you forget which is expected and which is actual!

CHAPTER 1

API

class `expecter.`**expect**(*actual*)

All assertions are written using `expect`. Usually, it's applied to the value you're making an assertion about:

```
>>> expect(5) > 4
expect(5)
>>> expect(4) > 4
Traceback (most recent call last):
...
AssertionError: Expected something greater than 4 but got 4
```

This works for comparisons as you'd expect:

==, !=, <, >, <=, >=

Note that expect() *always* goes around the actual value: the value you're making an assertion about.

There are other, non-binary expectations available. They're documented below.

contains(*other*)

Ensure that `other` is in the actual value (like `assert other in actual`).

does_not_contain(*other*)

Opposite of `contains`

isinstance(*expected_cls*)

Ensures that the actual value is of type `expected_cls` (like `assert isinstance(actual, MyClass)`).

static raises(*expected_cls=<type 'exceptions.Exception'>*, *message=None*)

Ensure that an exception is raised. E.g.,

```
with expect.raises(MyCustomError):
    func_that_raises_error()
```

is equivalent to:

```
try:
    func_that_raises_error()
    raise AssertionError('Error not raised!')
except MyCustomError:
    pass
```

`expecter.`**add_expectation**(*predicate*)

Add a custom expectation. After being added, custom expectations can be used as if they were built-in:

```
>>> def is_long(x): return len(x) > 5
>>> add_expectation(is_long)
>>> expect('looooooong').is_long()
>>> expect('short').is_long()
Traceback (most recent call last):
...
AssertionError: Expected that 'short' is_long, but it isn't
```

The name of the expectation is taken from the name of the function (as shown above).

expecter.**clear_expectations**()
Remove all custom expectations

e