PyHamcrest Documentation

Release 1.8.5

hamcrest.org

July 19, 2015

Contents

Contents:

CHAPTER 1

PyHamcrest Tutorial

1.1 Introduction

PyHamcrest is a framework for writing matcher objects, allowing you to declaratively define "match" rules. There are a number of situations where matchers are invaluable, such as UI validation, or data filtering, but it is in the area of writing flexible tests that matchers are most commonly used. This tutorial shows you how to use PyHamcrest for unit testing.

When writing tests it is sometimes difficult to get the balance right between overspecifying the test (and making it brittle to changes), and not specifying enough (making the test less valuable since it continues to pass even when the thing being tested is broken). Having a tool that allows you to pick out precisely the aspect under test and describe the values it should have, to a controlled level of precision, helps greatly in writing tests that are "just right." Such tests fail when the behavior of the aspect under test deviates from the expected behavior, yet continue to pass when minor, unrelated changes to the behaviour are made.

1.2 My first PyHamcrest test

We'll start by writing a very simple PyUnit test, but instead of using PyUnit's `assertEqual` method, we'll use PyHamcrest's `assert_that` construct and the standard set of matchers:

```
from hamcrest import *
import unittest

class BiscuitTest(unittest.TestCase):
    def testEquals(self):
        theBiscuit = Biscuit('Ginger')
        myBiscuit = Biscuit('Ginger')
        assert_that(theBiscuit, equal_to(myBiscuit))

if __name__ == '__main__':
    unittest.main()
```

The `assert_that` function is a stylized sentence for making a test assertion. In this example, the subject of the assertion is the object `theBiscuit`, which is the first method parameter. The second method parameter is a matcher for `Biscuit` objects, here a matcher that checks one object is equal to another using the Python == operator. The test passes since the `Biscuit` class defines an `__eq__` method.

If you have more than one assertion in your test you can include an identifier for the tested value in the assertion:

```
assert_that(theBiscuit.getChocolateChipCount(), equal_to(10), 'chocolate chips')
assert_that(theBiscuit.getHazelnutCount(), equal_to(3), 'hazelnuts')
```

As a convenience, `assert_that` can also be used to verify a boolean condition:

```
assert_that(theBiscuit.isCooked(), 'cooked')
```

This is equivalent to the `assert_` method of `unittest.TestCase`, but because it's a standalone function, it offers greater flexibility in test writing.

1.3 Asserting Exceptions

There's a utility function and matcher available to help you test that your code has the expected behavior in situations where it should raise an exception. The *calling* function wraps a callable, and then allows you to set arguments to be used in a call to the wrapped callable. This, together with the *raises* matcher lets you assert that calling a method with certain arguments causes an exception to be thrown. It is also possible to provide a regular expression pattern to the *raises* matcher allowing you assure that the right issue was found:

```
assert_that(calling(parse, bad_data), raises(ValueError))

assert_that(calling(translate).with_(curse_words), raises(LanguageError, "\w+very naughty"))

assert_that(broken_function, raises(Exception))

# This will fail and complain that 23 is not callable
# assert_that(23, raises(IOError))
```

1.4 Predefined matchers

PyHamcrest comes with a library of useful matchers:

- Object
  - *equal_to* - match equal object
  - *has_length* - match `len(item)`
  - *has_property* - match value of property with given name
  - ***has_properties*** **- match an** object that has all of the given properties.
  - *has_string* - match `str(item)`
  - *instance_of* - match object type
  - *none*, *not_none* - match `None`, or not `None`
  - *same_instance* - match same object
  - *calling*, *raises* - wrap a method call and assert that it raises an exception
- Number
  - *close_to* - match number close to a given value
  - *greater_than*, *greater_than_or_equal_to*, *less_than*, *less_than_or_equal_to* - match numeric ordering
- Text

  - *contains_string* - match part of a string
  - *ends_with* - match the end of a string
  - *equal_to_ignoring_case* - match the complete string but ignore case
  - *equal_to_ignoring_whitespace* - match the complete string but ignore extra whitespace
  - *starts_with* - match the beginning of a string
  - *string_contains_in_order* - match parts of a string, in relative order
- Logical
  - *all_of* - `and` together all matchers
  - *any_of* - `or` together all matchers
  - *anything* - match anything, useful in composite matchers when you don't care about a particular value
  - *is_not* - negate the matcher
- Sequence
  - *contains* - exactly match the entire sequence
  - *contains_inanyorder* - match the entire sequence, but in any order
  - *has_item* - match if given item appears in the sequence
  - *has_items* - match if all given items appear in the list, in any order
  - *is_in* - match if item appears in the given sequence
  - *only_contains* - match if sequence's items appear in given list
  - *empty* - match if the sequence is empty
- Dictionary
  - *has_entries* - match dictionary with list of key-value pairs
  - *has_entry* - match dictionary containing a key-value pair
  - *has_key* - match dictionary with a key
  - *has_value* - match dictionary with a value
- Decorator
  - *described_as* - give the matcher a custom failure description
  - *is_* - decorator to improve readability - see *Syntactic sugar*, below

The arguments for many of these matchers accept not just a matching value, but another matcher, so matchers can be composed for greater flexibility. For example, `only_contains(less_than(5))` will match any sequence where every item is less than 5.

1.5 Syntactic sugar

PyHamcrest strives to make your tests as readable as possible. For example, the *is_* matcher is a wrapper that doesn't add any extra behavior to the underlying matcher. The following assertions are all equivalent:

```
assert_that(theBiscuit, equal_to(myBiscuit))
assert_that(theBiscuit, is_(equal_to(myBiscuit)))
assert_that(theBiscuit, is_(myBiscuit))
```

The last form is allowed since `is_(value)` wraps most non-matcher arguments with *equal_to*. But if the argument is a type, it is wrapped with *instance_of*, so the following are also equivalent:

```
assert_that(theBiscuit, instance_of(Biscuit))
assert_that(theBiscuit, is_(instance_of(Biscuit)))
assert_that(theBiscuit, is_(Biscuit))
```

(Note that PyHamcrest's `is_` matcher is unrelated to Python's `is` operator. The matcher for object identity is *same_instance*.)

CHAPTER 2

Writing Custom Matchers

PyHamcrest comes bundled with lots of useful matchers, but you'll probably find that you need to create your own from time to time to fit your testing needs. This commonly occurs when you find a fragment of code that tests the same set of properties over and over again (and in different tests), and you want to bundle the fragment into a single assertion. By writing your own matcher you'll eliminate code duplication and make your tests more readable!

Let's write our own matcher for testing if a calendar date falls on a Saturday. This is the test we want to write:

```
def testDateIsOnASaturday(self):
    d = datetime.date(2008, 04, 26)
    assert_that(d, is_(on_a_saturday()))
```

And here's the implementation:

```
from hamcrest.core.base_matcher import BaseMatcher
from hamcrest.core.helpers.hasmethod import hasmethod

class IsGivenDayOfWeek(BaseMatcher):

    def __init__(self, day):
        self.day = day  # Monday is 0, Sunday is 6

    def _matches(self, item):
        if not hasmethod(item, 'weekday'):
            return False
        return item.weekday() == self.day

    def describe_to(self, description):
        day_as_string = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                         'Friday', 'Saturday', 'Sunday']
        description.append_text('calendar date falling on ')    \
                   .append_text(day_as_string[self.day])

def on_a_saturday():
    return IsGivenDayOfWeek(5)
```

For our Matcher implementation we implement the *_matches* method - which calls the `weekday` method after confirming that the argument (which may not be a date) has such a method - and the *describe_to* method - which is used to produce a failure message when a test fails. Here's an example of how the failure message looks:

```
assert_that(datetime.date(2008, 04, 06), is_(on_a_saturday()))
```

fails with the message:

```
AssertionError:
Expected: is calendar date falling on Saturday
     got: <2008-04-06>
```

Let's say this matcher is saved in a module named `isgivendayofweek`. We could use it in our test by importing the factory function `on_a_saturday`:

```
from hamcrest import *
import unittest
from isgivendayofweek import on_a_saturday

class DateTest(unittest.TestCase):
    def testDateIsOnASaturday(self):
        d = datetime.date(2008, 04, 26)
        assert_that(d, is_(on_a_saturday()))

if __name__ == '__main__':
    unittest.main()
```

Even though the `on_a_saturday` function creates a new matcher each time it is called, you should not assume this is the only usage pattern for your matcher. Therefore you should make sure your matcher is stateless, so a single instance can be reused between matches.

CHAPTER 3

Matcher Library

Library of Matcher implementations.

3.1 Object Matchers

Matchers that inspect objects.

3.1.1 equal_to

hamcrest.core.core.isequal.**equal_to**(*obj*)

Matches if object is equal to a given object.

Parameters **obj** – The object to compare against as the expected value.

This matcher compares the evaluated object to `obj` for equality.

3.1.2 has_length

hamcrest.library.object.haslength.**has_length**(*match*)

Matches if `len(item)` satisfies a given matcher.

Parameters **match** – The matcher to satisfy, or an expected value for *equal_to* matching.

This matcher invokes the `len` function on the evaluated object to get its length, passing the result to a given matcher for evaluation.

If the `match` argument is not a matcher, it is implicitly wrapped in an *equal_to* matcher to check for :equality.

Examples:

```
has_length(greater_than(6))
has_length(5)
```

3.1.3 has_string

hamcrest.library.object.hasstring.**has_string**(*match*)

Matches if `str(item)` satisfies a given matcher.

Parameters **match** – The matcher to satisfy, or an expected value for *equal_to* matching.

This matcher invokes the `str` function on the evaluated object to get its length, passing the result to a given matcher for evaluation. If the `match` argument is not a matcher, it is implicitly wrapped in an *`equal_to`* matcher to check for equality.

Examples:

```
has_string(starts_with('foo'))
has_string('bar')
```

3.1.4 has_properties/has_property

hamcrest.library.object.hasproperty.**has_properties**(**keys_valuematchers*, ***kv_args*)

Matches if an object has properties satisfying all of a dictionary of string property names and corresponding value matchers.

Parameters **matcher_dict** – A dictionary mapping keys to associated value matchers, or to expected values for *`equal_to`* matching.

Note that the keys must be actual keys, not matchers. Any value argument that is not a matcher is implicitly wrapped in an *`equal_to`* matcher to check for equality.

Examples:

```
has_properties({'foo':equal_to(1), 'bar':equal_to(2)})
has_properties({'foo':1, 'bar':2})
```

`has_properties` also accepts a list of keyword arguments:

hamcrest.library.object.hasproperty.**has_properties**(*keyword1=value_matcher1*[, *keyword2=value_matcher2*[, ...]])

Parameters

- **keyword1** – A keyword to look up.
- **valueMatcher1** – The matcher to satisfy for the value, or an expected value for *`equal_to`* matching.

Examples:

```
has_properties(foo=equal_to(1), bar=equal_to(2))
has_properties(foo=1, bar=2)
```

Finally, `has_properties` also accepts a list of alternating keys and their value matchers:

hamcrest.library.object.hasproperty.**has_properties**(*key1*, *value_matcher1*[, ...])

Parameters

- **key1** – A key (not a matcher) to look up.
- **valueMatcher1** – The matcher to satisfy for the value, or an expected value for *`equal_to`* matching.

Examples:

```
has_properties('foo', equal_to(1), 'bar', equal_to(2))
has_properties('foo', 1, 'bar', 2)
```

hamcrest.library.object.hasproperty.**has_property**(*name*, *match=None*)
Matches if object has a property with a given name whose value satisfies a given matcher.

Parameters

- **name** – The name of the property.
- **match** – Optional matcher to satisfy.

This matcher determines if the evaluated object has a property with a given name. If no such property is found, `has_property` is not satisfied.

If the property is found, its value is passed to a given matcher for evaluation. If the `match` argument is not a matcher, it is implicitly wrapped in an *`equal_to`* matcher to check for equality.

If the `match` argument is not provided, the *`anything`* matcher is used so that `has_property` is satisfied if a matching property is found.

Examples:

```
has_property('name', starts_with('J'))
has_property('name', 'Jon')
has_property('name')
```

3.1.5 instance_of

hamcrest.core.core.isinstanceof.**instance_of**(*atype*)
Matches if object is an instance of, or inherits from, a given type.

Parameters **atype** – The type to compare against as the expected type or a tuple of types.

This matcher checks whether the evaluated object is an instance of `atype` or an instance of any class that inherits from `atype`.

Example:

```
instance_of(str)
```

3.1.6 none, not_none

hamcrest.core.core.isnone.**none**()
Matches if object is `None`.

hamcrest.core.core.isnone.**not_none**()
Matches if object is not `None`.

3.1.7 same_instance

hamcrest.core.core.issame.**same_instance**(*obj*)
Matches if evaluated object is the same instance as a given object.

Parameters **obj** – The object to compare against as the expected value.

This matcher invokes the `is` identity operator to determine if the evaluated object is the the same object as `obj`.

3.1.8 calling, raises

hamcrest.core.core.raises.**calling**(*func*)
Wrapper for function call that delays the actual execution so that *raises* matcher can catch any thrown exception.

Parameters **func** – The function or method to be called

The arguments can be provided with a call to the `with_args` function on the returned object:

```
calling(my_method).with_args(arguments, and_='keywords')
```

hamcrest.core.core.raises.**raises**(*exception*, *pattern=None*)
Matches if the called function raised the expected exception.

Parameters

- **exception** – The class of the expected exception
- **pattern** – Optional regular expression to match exception message.

Expects the actual to be wrapped by using *calling*, or a callable taking no arguments. Optional argument pattern should be a string containing a regular expression. If provided, the string representation of the actual exception - e.g. `str(actual)` - must match pattern.

Examples:

```
assert_that(calling(int).with_args('q'), raises(TypeError))
assert_that(calling(parse, broken_input), raises(ValueError))
```

3.2 Number Matchers

Matchers that perform numeric comparisons.

3.2.1 close_to

hamcrest.library.number.iscloseto.**close_to**(*value*, *delta*)
Matches if object is a number close to a given value, within a given delta.

Parameters

- **value** – The value to compare against as the expected value.
- **delta** – The maximum delta between the values for which the numbers are considered close.

This matcher compares the evaluated object against `value` to see if the difference is within a positive `delta`.

Example:

```
close_to(3.0, 0.25)
```

hamcrest.library.number.iscloseto.**isnumeric**(*value*)
Confirm that 'value' can be treated numerically; duck-test accordingly

3.2.2 greater_than, greater_than_or_equal_to, less_than, less_than_or_equal_to

hamcrest.library.number.ordering_comparison.**greater_than**(*value*)
Matches if object is greater than a given value.

Parameters **value** – The value to compare against.

hamcrest.library.number.ordering_comparison.**greater_than_or_equal_to**(*value*)
Matches if object is greater than or equal to a given value.

Parameters **value** – The value to compare against.

hamcrest.library.number.ordering_comparison.**less_than**(*value*)
Matches if object is less than a given value.

Parameters **value** – The value to compare against.

hamcrest.library.number.ordering_comparison.**less_than_or_equal_to**(*value*)
Matches if object is less than or equal to a given value.

Parameters **value** – The value to compare against.

3.3 Text Matchers

Matchers that perform text comparisons.

3.3.1 contains_string

hamcrest.library.text.stringcontains.**contains_string**(*substring*)
Matches if object is a string containing a given string.

Parameters **string** – The string to search for.

This matcher first checks whether the evaluated object is a string. If so, it checks whether it contains `string`.

Example:

```
contains_string("def")
```

will match "abcdefg".

3.3.2 ends_with

hamcrest.library.text.stringendswith.**ends_with**(*string*)
Matches if object is a string ending with a given string.

Parameters **string** – The string to search for.

This matcher first checks whether the evaluated object is a string. If so, it checks if `string` matches the ending characters of the evaluated object.

Example:

```
ends_with("bar")
```

will match "foobar".

3.3.3 equal_to_ignoring_case

hamcrest.library.text.isequal_ignoring_case.**equal_to_ignoring_case**(*string*)
Matches if object is a string equal to a given string, ignoring case differences.

Parameters **string** – The string to compare against as the expected value.

This matcher first checks whether the evaluated object is a string. If so, it compares it with `string`, ignoring differences of case.

Example:

```
equal_to_ignoring_case("hello world")
```

will match "heLLo WorlD".

3.3.4 equal_to_ignoring_whitespace

hamcrest.library.text.isequal_ignoring_whitespace.**equal_to_ignoring_whitespace**(*string*)
Matches if object is a string equal to a given string, ignoring differences in whitespace.

Parameters **string** – The string to compare against as the expected value.

This matcher first checks whether the evaluated object is a string. If so, it compares it with `string`, ignoring differences in runs of whitespace.

Example:

```
equal_to_ignoring_whitespace("hello world")
```

will match `"hello world"`.

3.3.5 matches_regexp

hamcrest.library.text.stringmatches.**matches_regexp**(*pattern*)
Matches if object is a string containing a match for a given regular expression.

Parameters **pattern** – The regular expression to search for.

This matcher first checks whether the evaluated object is a string. If so, it checks if the regular expression `pattern` matches anywhere within the evaluated object.

3.3.6 starts_with

hamcrest.library.text.stringstartswith.**starts_with**(*substring*)
Matches if object is a string starting with a given string.

Parameters **string** – The string to search for.

This matcher first checks whether the evaluated object is a string. If so, it checks if `string` matches the beginning characters of the evaluated object.

Example:

```
starts_with("foo")
```

will match "foobar".

3.3.7 string_contains_in_order

hamcrest.library.text.stringcontainsinorder.**string_contains_in_order**(*string1*[, *string2*[, ...]])

Matches if object is a string containing a given list of substrings in relative order.

Parameters **string1, ...** – A comma-separated list of strings.

This matcher first checks whether the evaluated object is a string. If so, it checks whether it contains a given list of strings, in relative order to each other. The searches are performed starting from the beginning of the evaluated string.

Example:

```
string_contains_in_order("bc", "fg", "jkl")
```

will match "abcdefghijklm".

3.4 Logical Matchers

Boolean logic using other matchers.

3.4.1 all_of

hamcrest.core.core.allof.**all_of**(*matcher1*[, *matcher2*[, ...]])

Matches if all of the given matchers evaluate to `True`.

Parameters **matcher1, ...** – A comma-separated list of matchers.

The matchers are evaluated from left to right using short-circuit evaluation, so evaluation stops as soon as a matcher returns `False`.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

3.4.2 any_of

hamcrest.core.core.anyof.**any_of**(*matcher1*[, *matcher2*[, ...]])

Matches if any of the given matchers evaluate to `True`.

Parameters **matcher1, ...** – A comma-separated list of matchers.

The matchers are evaluated from left to right using short-circuit evaluation, so evaluation stops as soon as a matcher returns `True`.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

3.4.3 anything

hamcrest.core.core.isanything.**anything**([*description*])

Matches anything.

Parameters **description** – Optional string used to describe this matcher.

This matcher always evaluates to `True`. Specify this in composite matchers when the value of a particular element is unimportant.

3.4.4 is_not

hamcrest.core.core.isnot.**is_not**(*match*)

Inverts the given matcher to its logical negation.

Parameters **match** – The matcher to negate.

This matcher compares the evaluated object to the negation of the given matcher. If the `match` argument is not a matcher, it is implicitly wrapped in an *equal_to* matcher to check for equality, and thus matches for inequality.

Examples:

```
assert_that(cheese, is_not(equal_to(smelly)))
assert_that(cheese, is_not(smelly))
```

hamcrest.core.core.isnot.**not_**(*match*)

Alias of *is_not* for better readability of negations.

Examples:

```
assert_that(alist, not_(has_item(item)))
```

3.5 Sequence Matchers

Matchers of sequences.

3.5.1 contains

hamcrest.library.collection.issequence_containinginorder.**contains**(*match1*[, *match2*[, ...]])

Matches if sequence's elements satisfy a given list of matchers, in order.

Parameters **match1, ...** – A comma-separated list of matchers.

This matcher iterates the evaluated sequence and a given list of matchers, seeing if each element satisfies its corresponding matcher.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

3.5.2 contains_inanyorder

hamcrest.library.collection.issequence_containinginanyorder.**contains_inanyorder**(*match1*[, *match2*[, ...]])

Matches if sequences's elements, in any order, satisfy a given list of matchers.

Parameters **match1, ...** – A comma-separated list of matchers.

This matcher iterates the evaluated sequence, seeing if each element satisfies any of the given matchers. The matchers are tried from left to right, and when a satisfied matcher is found, it is no longer a candidate for the remaining elements. If a one-to-one correspondence is established between elements and matchers, `contains_inanyorder` is satisfied.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

3.5.3 has_item, has_items

hamcrest.library.collection.issequence_containing.**has_item**(*match*)

Matches if any element of sequence satisfies a given matcher.

Parameters **match** – The matcher to satisfy, or an expected value for *equal_to* matching.

This matcher iterates the evaluated sequence, searching for any element that satisfies a given matcher. If a matching element is found, `has_item` is satisfied.

If the `match` argument is not a matcher, it is implicitly wrapped in an *equal_to* matcher to check for equality.

hamcrest.library.collection.issequence_containing.**has_items**(*match1*[, *match2*[, ...]])

Matches if all of the given matchers are satisfied by any elements of the sequence.

Parameters **match1, ...** – A comma-separated list of matchers.

This matcher iterates the given matchers, searching for any elements in the evaluated sequence that satisfy them. If each matcher is satisfied, then `has_items` is satisfied.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

3.5.4 is_in

hamcrest.library.collection.isin.**is_in**(*sequence*)

Matches if evaluated object is present in a given sequence.

Parameters **sequence** – The sequence to search.

This matcher invokes the `in` membership operator to determine if the evaluated object is a member of the sequence.

3.5.5 only_contains

hamcrest.library.collection.issequence_onlycontaining.**only_contains**(*match1*[, *match2*[, ...]])

Matches if each element of sequence satisfies any of the given matchers.

Parameters **match1, ...** – A comma-separated list of matchers.

This matcher iterates the evaluated sequence, confirming whether each element satisfies any of the given matchers.

Example:

```
only_contains(less_than(4))
```

will match `[3,1,2]`.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

3.5.6 empty

hamcrest.library.collection.is_empty.**empty**()
This matcher matches any collection-like object that responds to the __len__ method, and has a length of 0.

3.6 Dictionary Matchers

Matchers of dictionaries.

3.6.1 has_entries

hamcrest.library.collection.isdict_containingentries.**has_entries**(*matcher_dict*)
Matches if dictionary contains entries satisfying a dictionary of keys and corresponding value matchers.

Parameters **matcher_dict** – A dictionary mapping keys to associated value matchers, or to expected values for *equal_to* matching.

Note that the keys must be actual keys, not matchers. Any value argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

Examples:

```
has_entries({'foo':equal_to(1), 'bar':equal_to(2)})
has_entries({'foo':1, 'bar':2})
```

has_entries also accepts a list of keyword arguments:

hamcrest.library.collection.isdict_containingentries.**has_entries**(*keyword1=value_matcher1*[, *keyword2=value_matcher2*[, ...]])

Parameters

- **keyword1** – A keyword to look up.
- **valueMatcher1** – The matcher to satisfy for the value, or an expected value for *equal_to* matching.

Examples:

```
has_entries(foo=equal_to(1), bar=equal_to(2))
has_entries(foo=1, bar=2)
```

Finally, has_entries also accepts a list of alternating keys and their value matchers:

hamcrest.library.collection.isdict_containingentries.**has_entries**(*key1*, *value_matcher1*[, ...])

Parameters

- **key1** – A key (not a matcher) to look up.
- **valueMatcher1** – The matcher to satisfy for the value, or an expected value for *equal_to* matching.

Examples:

```
has_entries('foo', equal_to(1), 'bar', equal_to(2))
has_entries('foo', 1, 'bar', 2)
```

3.6.2 has_entry

hamcrest.library.collection.isdict_containing.**has_entry**(*key_match*, *value_match*)

Matches if dictionary contains key-value entry satisfying a given pair of matchers.

Parameters

- **key_match** – The matcher to satisfy for the key, or an expected value for *equal_to* matching.
- **value_match** – The matcher to satisfy for the value, or an expected value for *equal_to* matching.

This matcher iterates the evaluated dictionary, searching for any key-value entry that satisfies `key_match` and `value_match`. If a matching entry is found, `has_entry` is satisfied.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

Examples:

```
has_entry(equal_to('foo'), equal_to(1))
has_entry('foo', 1)
```

3.6.3 has_key

hamcrest.library.collection.isdict_containingkey.**has_key**(*key_match*)

Matches if dictionary contains an entry whose key satisfies a given matcher.

Parameters **key_match** – The matcher to satisfy for the key, or an expected value for *equal_to* matching.

This matcher iterates the evaluated dictionary, searching for any key-value entry whose key satisfies the given matcher. If a matching entry is found, `has_key` is satisfied.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

Examples:

```
has_key(equal_to('foo'))
has_key('foo')
```

3.6.4 has_value

hamcrest.library.collection.isdict_containingvalue.**has_value**(*value*)

Matches if dictionary contains an entry whose value satisfies a given matcher.

Parameters **value_match** – The matcher to satisfy for the value, or an expected value for *equal_to* matching.

This matcher iterates the evaluated dictionary, searching for any key-value entry whose value satisfies the given matcher. If a matching entry is found, `has_value` is satisfied.

Any argument that is not a matcher is implicitly wrapped in an *equal_to* matcher to check for equality.

Examples:

```
has_value(equal_to('bar'))
has_value('bar')
```

3.7 Decorator Matchers

Matchers that decorate other matchers for better expression.

3.7.1 described_as

hamcrest.core.core.described_as.**described_as**(*description*, *matcher*[, *value1*[, ...]])

Adds custom failure description to a given matcher.

Parameters

- **description** – Overrides the matcher's description.
- **matcher** – The matcher to satisfy.
- **value1,...** – Optional comma-separated list of substitution values.

The description may contain substitution placeholders %0, %1, etc. These will be replaced by any values that follow the matcher.

3.7.2 is_

hamcrest.core.core.is_.**is_**(*x*)

Decorates another matcher, or provides shortcuts to the frequently used `is(equal_to(x))` and `is(instance_of(x))`.

Parameters **x** – The matcher to satisfy, or a type for *instance_of* matching, or an expected value for *equal_to* matching.

This matcher compares the evaluated object to the given matcher.

Note: PyHamcrest's `is_` matcher is unrelated to Python's `is` operator. The matcher for object identity is *same_instance*.

If the `x` argument is a matcher, its behavior is retained, but the test may be more expressive. For example:

```
assert_that(value, less_than(5))
assert_that(value, is_(less_than(5)))
```

If the `x` argument is a type, it is wrapped in an *instance_of* matcher. This makes the following statements equivalent:

```
assert_that(cheese, instance_of(Cheddar))
assert_that(cheese, is_(instance_of(Cheddar)))
assert_that(cheese, is_(Cheddar))
```

Otherwise, if the `x` argument is not a matcher, it is wrapped in an *equal_to* matcher. This makes the following statements equivalent:

```
    assert_that(cheese, equal_to(smelly))
    assert_that(cheese, is_(equal_to(smelly)))
    assert_that(cheese, is_(smelly))
```

Choose the style that makes your expression most readable. This will vary depending on context.

CHAPTER 4

Integration with PyUnit and Other Libraries

4.1 assert_that

hamcrest.core.assert_that.**assert_that**(*actual*, *matcher*[, *reason*])

Asserts that actual value satisfies matcher. (Can also assert plain boolean condition.)

Parameters

- **actual** – The object to evaluate as the actual value.
- **matcher** – The matcher to satisfy as the expected condition.
- **reason** – Optional explanation to include in failure description.

assert_that passes the actual value to the matcher for evaluation. If the matcher is not satisfied, an exception is thrown describing the mismatch.

assert_that is designed to integrate well with PyUnit and other unit testing frameworks. The exception raised for an unmet assertion is an AssertionError, which PyUnit reports as a test failure.

With a different set of parameters, assert_that can also verify a boolean condition:

hamcrest.core.assert_that.**assert_that**(*assertion*[, *reason*])

Parameters

- **assertion** – Boolean condition to verify.
- **reason** – Optional explanation to include in failure description.

This is equivalent to the assertTrue method of unittest.TestCase, but offers greater flexibility in test writing by being a standalone function.

4.2 match_equality

hamcrest.library.integration.match_equality.**match_equality**(*matcher*)

Wraps a matcher to define equality in terms of satisfying the matcher.

match_equality allows Hamcrest matchers to be used in libraries that are not Hamcrest-aware. They might use the equality operator:

```
assert match_equality(matcher) == object
```

Or they might provide a method that uses equality for its test:

```
library.method_that_tests_eq(match_equality(matcher))
```

One concrete example is integrating with the `assert_called_with` methods in Michael Foord's mock library.

CHAPTER 5

Core API

5.1 Helpers

Utilities for writing Matchers

5.1.1 hasmethod

hamcrest.core.helpers.hasmethod.**hasmethod**(*obj*, *methodname*)
Does `obj` have a method named `methodname`?

5.1.2 wrap_matcher

hamcrest.core.helpers.wrap_matcher.**wrap_matcher**(*x*)
Wraps argument in a matcher, if necessary.

Returns the argument as-is if it is already a matcher, otherwise wrapped in an *equal_to* matcher.

5.2 BaseDescription

class hamcrest.core.base_description.**BaseDescription**
Bases: *hamcrest.core.description.Description*

Base class for all *Description* implementations.

append(*string*)
Append the string to the description.

5.3 BaseMatcher

class hamcrest.core.base_matcher.**BaseMatcher**
Bases: *hamcrest.core.matcher.Matcher*

Base class for all *Matcher* implementations.

Most implementations can just implement *_matches*, leaving the handling of any mismatch description to the `matches` method. But if it makes more sense to generate the mismatch description during the matching, override *matches* instead.

_matches(*item*)

describe_mismatch(*item*, *mismatch_description*)

matches(*item*, *mismatch_description=None*)

5.4 Description

class hamcrest.core.description.**Description**
Bases: object

A description of a *Matcher*.

A *Matcher* will describe itself to a description which can later be used for reporting.

append_description_of(*value*)
Appends description of given value to this description.

If the value implements *describe_to*, then it will be used.

Returns self, for chaining

append_list(*start*, *separator*, *end*, *list*)
Appends a list of objects to the description.

Parameters

- **start** – String that will begin the list description.
- **separator** – String that will separate each object in the description.
- **end** – String that will end the list description.
- **list** – List of objects to be described.

Returns self, for chaining

append_text(*text*)
Appends some plain text to the description.

Returns self, for chaining

append_value(*value*)
Appends an arbitary value to the description.

Deprecated: Call *append_description_of* instead.

Returns self, for chaining

5.5 Matcher

class hamcrest.core.matcher.**Matcher**
Bases: *hamcrest.core.selfdescribing.SelfDescribing*

A matcher over acceptable values.

A matcher is able to describe itself to give feedback when it fails.

Matcher implementations should *not* directly implement this protocol. Instead, *extend* the *BaseMatcher* class, which will ensure that the *Matcher* API can grow to support new features and remain compatible with all *Matcher* implementations.

describe_mismatch(*item*, *mismatch_description*)
Generates a description of why the matcher has not accepted the item.

The description will be part of a larger description of why a matching failed, so it should be concise.

This method assumes that `matches(item)` is `False`, but will not check this.

Parameters

- **item** – The item that the *Matcher* has rejected.
- **mismatch_description** – The description to be built or appended to.

matches(*item*, *mismatch_description=None*)
Evaluates the matcher for argument item.

If a mismatch is detected and argument `mismatch_description` is provided, it will generate a description of why the matcher has not accepted the item.

Parameters **item** – The object against which the matcher is evaluated.

Returns `True` if `item` matches, otherwise `False`.

5.6 SelfDescribing

class hamcrest.core.selfdescribing.**SelfDescribing**
Bases: `object`

The ability of an object to describe itself.

describe_to(*description*)
Generates a description of the object.

The description may be part of a description of a larger object of which this is just a component, so it should be worded appropriately.

Parameters **description** – The description to be built or appended to.

5.7 SelfDescribingValue

class hamcrest.core.selfdescribingvalue.**SelfDescribingValue**(*value*)
Bases: *hamcrest.core.selfdescribing.SelfDescribing*

Wrap any value in a `SelfDescribingValue` to satisfy the *SelfDescribing* interface.

Deprecated: No need for this class now that *append_description_of* handles any type of value.

describe_to(*description*)
Generates a description of the value.

5.8 StringDescription

class hamcrest.core.string_description.**StringDescription**
Bases: *hamcrest.core.base_description.BaseDescription*

A *Description* that is stored as a string.

hamcrest.core.string_description.**tostring**(*selfdescribing*)
Returns the description of a *SelfDescribing* object as a string.

Parameters **selfdescribing** – The object to be described.

Returns The description of the object.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

h

Symbols

A

B

C

D

E

G

H

I

L

M

N

O

R

S

T

W