Pycon Tutorial - Python 102

Simon Lau

18 Jun 2014

1. Administration
2. Session 1
3. Session 2

Administration

Internet and downloads

PENDING

Author

Simon Lau

Lecturer

School of Infocomm

Republic Polytechnic

simon_lau@rp.edu.sg

Agenda

- Session 1
  - Numbers and Strings
  - String and conditions
  - Functions
  - Namespaces
  - Introspection
- Tea Break at 3:45pm
- Session 2
  - Files
  - Classes
  - Iterators
  - Exceptions
  - Testing

What are you most interested in?

Session 1

Numbers and Strings

Numeric Types (Built-in)

```
a = 3       # int
b = 3.5     # float
c = 3 + 2j  # complex (real + imaginary)

x = 127     # base 10
y = 0177    # base 8 (octal)
z = 0x7F    # base 16 (hexadecimal)
```

Numeric Types (Library)

Fraction[a]

```
from fractions import Fraction
d = Fraction(16, -10)
```

[a] http://www.doughellmann.com/PyMOTW/fractions/index.html

Decimal[a]

```
from decimal import Decimal
e = Decimal('0.1')
print e + e + e - Decimal('0.3')

print 0.1 + 0.1 + 0.1 - 0.3
```

[a] http://www.doughellmann.com/PyMOTW/decimal/index.html

String Types

Strings(str) are immutable sequences

```
a = 'Hello'           # Single quotes
b = "World"           # Double quotes
c = 'Two \n Lines'    # Control characters
d = r'One \n Line'    # Raw String (no control chars)
e = u'Extra'          # Unicode (default for Python 3.x)
f = '''
Line One
Line Two
'''                   # Triple-quoted strings
```

Typecasting

- Use str(x), int(x), float(x) to convert data from one type to another
  - Recall that "1" is not the same as 1
  - Some mathematical operations only work on data with the same type
  - If necessary and appropriate, convert the data to the correct type before executing the mathematical operations

```
a = 1
b = str(a)     # b = ’1’
c = float(a)   # c = 1.0
d = ’33’
e = int(d)     # e = 33
f = d + b      # f = ’33’ + ’1’ = ’331’
g = e + a      # g = 33 + 1 = 34
```

String (and list) functions

```
s = ’Republic’
t = ’Polytechnic’

a = ’p’ in s      # True
b = ’p’ not in s  # False
c = s + t         # ’RepublicPolytechnic’
d = s * 2         # ’RepublicRepublic’
e = len(s)        # 8
f = min(s)        # ’R’ (based on ASCII ordering)
g = max(s)        # ’u’ (based on ASCII ordering)
h = t.count(’c’)  # 2
i = t.index(’c’)  # 6
```

Slicing strings and lists

```
s = 'Republic'

n = s[-1]    # 'c' (negative index start from the back)
p = s[2]     # 'p'
q = s[2:5]   # 'pub' (slice from start:end)
r = s[2:6:2] # 'pb' (slice from start:end:step)
```

Short Exercise

Given a str, return a *"rotated left 2"* version where the first 2 char are moved to the end. The str length will be at least 2. [1]

```
left2('Hello') # 'lloHe'
left2('java')  # 'vaja'
left2('Hi')    # 'Hi'
```

[1] http://codingbat.com/prob/p160545

Going through lists

Python for loop

```
my_list = [’a’, 1, True]
for item in my_list:
    print(item)
```

Java for-each loop

```
String myList[] = {"Hello", "World", "Pycon"};
for (item : myList) {
    System.out.println(myList)
}
```

List Comprehension

Long form (Before Python 2.0)

```
x = [1, 5, 6, 2, 8]
y = []
for item in x:
    if item > 3:      # selection
        z = item + 1  # processing
        y.append(z)
```

Short form (After Python 2.0)

```
x = [1, 5, 6, 2, 8]
y = [item + 1 for item in x if item > 3]
```

Range

- range() is a function that can be used to generate a series of numbers
- range(**start**, **end**, **step**) will produce a series of number
    - Starting from **start** (default to 0)
    - Ending at **end - 1**
    - And skipping number based on the **step** (default to 1)

```
a = range(5)          # [0, 1, 2, 3, 4]
b = range(10)         # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
c = range(2, 5, 1)    # [2, 3, 4]
d = range(2, 10, 2)   # [2, 4, 6, 8]
e = range(2, 10, 3)   # [2, 5, 8]
```

Xrange

- In Python 2.x
  - `range()` creates the entire `list` in memory
  - `xrange()` creates an **iterator** (not a `list`)
    - uses less memory if `list` is very large
- In Python 3.x
  - `range()` works like `xrange()` in Python 2.x
  - `xrange()` is removed

Loop Exercise

Given a `list` of `int`, return the number of `9`'s in the `list` [2]

```
array_count9([1, 2, 9])       # 1
array_count9([1, 9, 9])       # 2
array_count9([1, 9, 9, 3, 9]) # 3
```

[2] http://codingbat.com/prob/p166170

Useful Standard Library Modules for List

- array [3]
  - Manage sequences of fixed-type numerical data efficiently (From Python 1.4)
- heapq [4]
  - In-place heap sort algorithm (From Python 2.3)
- bisect [5]
  - Maintain list in sorted order (From Python 1.4)
- queue [6]
  - A thread-safe *FIFO* implementation (From Python 1.4)
- collection [7]
  - Counter (Bag which keeps count)
  - defaultdict (dict with default on creation)
  - Deque (list which is modifiable at either end)
  - namedtuple (tuple with named fields)
  - OrderedDict (dict which remember order)

[3] http://www.doughellmann.com/PyMOTW/array/index.html

Bisect Module

```
import bisect
import random

x = []
for i in range(10):
    r = random.randint(1, 100)
    bisect.insort(x, r)

print x
```

Counter Module

```
from collections import Counter

letters = [’x’, ’y’, ’x’, ’z’, ’y’, ’x’]

c = Counter(letters)
print c # Counter({’x’: 3, ’y’: 2, ’z’: 1})
```

Default Dictionary Module

```
from collections import defaultdict

def default_value():
    return 0

d = defaultdict(default_value)
d[’x’] = 5

print d[’x’]  # 5
print d[’y’]  # 0 (default value)
```

Named Tuple Module

```
from collections import namedtuple

sam = ('Sam', 'rp', 3.5)
print sam
print sam[2]    # GPA - 3.5

Student = namedtuple('Student', 'name poly gpa')
john = Student(name='John', poly='sp', gpa=3.0)
print john
print john.gpa # GPA - 3.0
```

OrderedDict Module

```
from collections import OrderedDict

print ’Regular dictionary:’
d = {}
d[’sp’] = ’Singapore Poly’
d[’np’] = ’Ngee Ann Poly’
d[’tp’] = ’Temasek Poly’

for k, v in d.items():
    print k, v

print ’\nOrderedDict:’
d = OrderedDict()
d[’sp’] = ’Singapore Poly’
d[’np’] = ’Ngee Ann Poly’
d[’tp’] = ’Temasek Poly’
```

String and conditions

Count and find

```
s = ’hello World’
g = s.capitalize()          # ’Hello world’
h = s.upper()               # ’HELLO WORLD’
i = s.lower()               # ’hello world’
j = s.replace(’o’, ’O’)     # ’hellO WOrld’

t = ’ hello world! ’
k = t.strip()               # ’hello world!’
```

Changing Strings

```
s = 'hello World'
g = s.capitalize()       # 'Hello World'
h = s.upper()            # 'HELLO WORLD'
i = s.lower()            # 'hello world'
j = s.replace('o', 'O')  # 'hellO WOrld'

t = ' hello world! '
k = t.strip()            # 'hello world!'
```

Split / Join

Split (String into list)

```
r = ’Hello-World-Bye-Pycon’
s = r.split(’-’)  # [’Hello’, ’World’, ’Bye’, ’Pycon’]
```

Join (List into String)

```
i = [’Hello’, ’World’, ’Bye’, ’Pycon’]
j = ’-’.join(i)  # Hello-World-Bye-Pycon
```

String formatting

Before Python 2.6

```
place = ’RP’
age = 10
output = ’%s is %d years old’ % (place, age)
```

After Python 2.6

```
place = ’RP’
age = 10
output = ’{0} is {1} years old’.format(place, age)
```

String exercise

- Given 2 str, a and b [8]
  - print a str of the form short+long+short
  - with the shorter str on the outside and the longer str on the inside
  - The str will not be the same length, but they may be empty (len 0)

```
combo_string('Hello', 'hi')  # print 'hiHellohi'
combo_string('hi', 'Hello')  # print 'hiHellohi'
combo_string('aaa', 'b')     # print 'baaab'
```

[8] http://codingbat.com/prob/p194053

Useful Standard Library Modules for String

- re [9]
    - Regular Expressions (From Python 1.5)
- textwrap [10]
    - Formatting text paragraphs (From Python 2.5)
- string [11]
    - Contains depreciated functions which are shifted to str methods
    - Templates (From Python 2.4)

[9] http://www.doughellmann.com/PyMOTW/re/index.html
[10] http://www.doughellmann.com/PyMOTW/textwrap/index.html
[11] http://www.doughellmann.com/PyMOTW/string/index.html

Regular Expressions

```
from re import search

patterns = [ ’this’, ’that’ ]
text = ’Does this text match the pattern?’

for pattern in patterns:
    print ’Looking for "%s" in "%s" ->’ % (pattern, text),

    if search(pattern,  text):
        print ’found a match!’
    else:
        print ’no match’
```

Functions

Function Syntax

```
def functionname(arg1, arg2, ...):
    '''
    docstring
    '''
    statement1
    statement2
    ...
```

Function Example

```
def greeting():
    ’’’
    Greeting for Simon @ Pycon
    ’’’
    print(’Hi, Simon’)
    print(’Welcome to Pycon!’)

greeting()  # call the function
```

Returning values

```
def greeting():
    s = ’Hi, Simon\nWelcome to Pycon!’
    return s

g = greeting()    # call the function and store the return
print(g.upper())  # change the return value
```

Passing parameters

```
def greeting(name, place):
    s = ’Hi, {0}\nWelcome to {1}!’.format(name, place)
    return s

g = greeting(’Simon’, ’Pycon’)
print(g)
```

Passing parameters by name

```
def greeting(name, place):
    s = 'Hi, {0}\nWelcome to {1}!'.format(name, place)
    return s

g = greeting(place='Pycon', name='Simon')
print(g)
```

Default param values

```
def greeting(name, place=’Nowhere’):
    s = ’Hi, {0}\nWelcome to {1}!’.format(name, place)
    return s

g = greeting(’Simon’)
print(g)
```

Arbitary argument list

```
def greeting(name, *args):
    others = '-'.join(args)
    s = 'Hi, {0}\nWelcome to {1}!'.format(name, others)
    return s

g = greeting('Simon', 'Pycon', 'in', 'RP')
print(g)  # Hi Simon. Welcome to Pycon-in-RP
```

Anonymous function generator lambda

- lambda is an anonymous (unnamed) function [12]
- Used primarily to write very short (one-line) functions

```
def greeting(name, place):
    s = 'Hi, {0}\nWelcome to {1}!'.format(name, place)
    return s

x = lambda n, p: 'Hi, {0}\nWelcome to {1}!'.format(n, p)
print(x('Simon', 'Pycon'))
```

[12] http://docs.python.org/tutorial/controlflow.html#lambda-forms

Main Function

- In Python a function called `main` doesn't have any special significance
- However, it is common practice to organize a program's main functionality in a function called `main` and call it with code similar to the following:

```
def main():
    pass  # the main code goes here

if __name__ == "__main__":
    main()
```

- When a Python program is executed directly
  - as opposed to being imported from another program
  - the special global variable `__name__` has the value `"__main__"`

Useful Standard Library Modules for Functions

- functools [13]
  - Tools for Manipulating Function (From Python 2.5)
- operator [14]
  - Functional interface to built-in operators (From Python 1.4)

[13] http://www.doughellmann.com/PyMOTW/functools/index.html

[14] http://www.doughellmann.com/PyMOTW/operator/index.html

Python Sorting

```
student_tuples = [
  (’john’, ’sp’, 4.0),
  (’jane’, ’np’, 3.5),
  (’dave’, ’rp’, 3.0),
]
student_tuples.sort()
print student_tuples

student_tuples.sort(key=lambda s: s[2]) # sort by gpa
print student_tuples
```

Operator Module

```
from operator import itemgetter

student_tuples.sort(key=itemgetter(2))
print student_tuples

# sort by poly then gpa
student_tuples.sort(key=itemgetter(1, 2))
print student_tuples
```

Namespaces

Import / namespaces

Import module

```
import random

x = random.randint(1, 6)
```

Import function

```
from random import randint

y = randint(1, 6)
```

Create your own library and namespace

Create util.py

```
def greeting(name, place):
    s = ’Hi, {0}\nWelcome to {1}!’.format(name, place)
    return s
```

Write main.py

```
from util import greeting

g = greeting(’Simon’, ’Pycon’)
print(g)
```

Importing from directories below your current

Create mod/greet.py

Same contents as `util.py`

Create `mod/__init__.py`

An empty file (Used to mark directory as module)

Write mainmod.py

```
from mod.greet import greeting

g = greeting('Simon', 'Pycon')
print(g)
```

Introspection

Asking for help()

```
>>> help

>>> help()

>>> help(’modules’)
```

Checking the sys module

```
import sys

print(sys.executable)

print(sys.platform)

print(sys.version)

print(sys.version_info)

print(sys.maxint)

print(sys.path)
```

Under the hood with dir()

Checking local scope

```
print(dir())
```

Checking modules

```
import sys
print(dir(sys))
```

Documentation strings

```
def greeting(name=’nobody’, place=’nowhere’):
    ’’’
    Function to greet someone at a place

    Keyword arguments:
    name -- the person to greet (default ’nobody’)
    place -- the place for the greeting (default ’nowhere’)
    ’’’
    s = ’Hi, {0}\nWelcome to {1}!’.format(name, place)
    return s

print(help(greeting))
```

Session 2

Files

Open, read whole file at once

Before Python 2.5

```
f = open(’pycon.txt’)
content = f.read()
print(content)
f.close()
```

After Python 2.5

```
with open(’pycon.txt’) as f:
    content = f.read()
    print(content)
# Note the f is automatically closed by here
```

File Modes

Actual API is `open(filename, mode)` [15]

Mode	Description
`r`	Read-only (**default** if omitted)
`w`	Write
`a`	Appending
`r+`	Read-write
`b`	Binary (for use on *Windows*)

[15] http://docs.python.org/tutorial/inputoutput.html#reading-and-writing-files

Read line by line

```
c = []
with open(’pycon.txt’) as f:
    for line in f:
        c.append(line)
print(c)
```

Writing line by line and close

```
info = [’Hello’, ’World’, ’Goodbye’, ’Pycon’]
with open(’simon.txt’, ’w’) as f:
    for item in info:
        f.write(item + ’\n’)
```

Read / write and seek

```
with open('line.txt', 'r+') as f:
    f.write('0123456789abcdef')
    f.seek(5)       # Go to the 6th byte in the file
    a = f.read(1)   # '5'
    f.seek(-3, 2)   # Go to the 3rd byte before the end
    b = f.read(1)   # 'd'
```

Useful Standard Library Modules for Files

- os.path [16]
  - Platform-independent manipulation of file names
- linecache [17]
  - Read text files efficiently (From Python 1.4)
- tempfile [18]
  - Create temporary filesystem resources (From Python 1.4)
- shutil [19]
  - High-level file operations (From Python 1.4)

[16] http://http://www.doughellmann.com/PyMOTW/ospath/index.html
[17] http://www.doughellmann.com/PyMOTW/linecache/index.html
[18] http://http://www.doughellmann.com/PyMOTW/tempfile/index.html
[19] http://www.doughellmann.com/PyMOTW/shutil/index.html

Shutil Module

```
from shutil import copyfile, move

copyfile('a.txt', 'acopy.txt')
move('a.txt', 'b.txt')
```

Classes

Constructing with __init__

```
class Human:
    def __init__(self):
        self.name = ’Simon’

h = Human()
print(h.name)
```

Class, instance and attributes

```
class Human:
    salutation = ’Mr’
    def __init__(self):
        self.name = ’Simon’

print(Human.salutation)

h = Human()
print(h.name)
print(h.salutation)
```

Self?

- Often, the first argument of a method is called `self`
- This is nothing more than a convention:
  - the name `self` has absolutely no special meaning to *Python*
  - Note, however, that by not following the convention your code may be less readable to other *Python* programmers

Overwritting __getitem__ and __setitem__

Making your class seem like a dict

```
class Human:
    def __getitem__(self, key):
        if key == ’name’:
            return self.name
    def __setitem__(self, key, value):
        if key == ’name’:
            self.name = value

h = Human()
h[’name’] = ’Simon’
print(h[’name’])
```

Linking up the class's iteritems to the list

Making your class seem like a list

```
class Human:
    def __init__(self, first, last):
        self.first = first
        self.last = last

    def __iter__(self):
        name = [self.first, self.last]
        for n in name:
            yield n

h = Human('Simon', 'Lau')
for i in h:
    print(i)
```

Inheritance Example

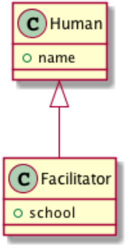

Figure : Single Inheritance

Inheritance Before Python 2.4

```
class Human():
    def __init__(self, name):
        self.name = name

class Facilitator(Human):
    def __init__(self, name, school='RP'):
        Human.__init__(self, name)
        self.school = school
```

Inheritance After Python 2.4

```
class Human(object):
    def __init__(self, name):
        self.name = name

class Facilitator(Human):
    def __init__(self, name, school=’RP’):
        super(Facilitator, self).__init__(name)
        self.school = school
```

Inheritance After Python 3.0

```
class Human(object):
    def __init__(self, name):
        self.name = name

class Facilitator(Human):
    def __init__(self, name, school=’RP’):
        super().__init__(name)
        self.school = school
```

@staticmethod

Before Python 2.4

```
class Human(object):
    @staticmethod
    def say(msg):
        return msg

print(Human.say(’Hi’))
```

After Python 2.4

```
class Human(object):
    @staticmethod
    def say(msg):
        return msg

print(Human.say(’Hi’))
```

@classmethod Before Python 2.4

```
class Human:
    def create(cls):
        instance = cls()
        instance.name = 'Simon'
        return instance
    create = classmethod(create)


class Facilitator(Human):
    def create(cls):
        instance = cls()
        instance.school = 'RP'
        return instance
    create = classmethod(create)
```

@classmethod After Python 2.4

```
class Human(object):
    @classmethod
    def create(cls):
        instance = cls()
        instance.name = ’Simon’
        return instance


class Facilitator(Human):
    @classmethod
    def create(cls):
        instance = cls()
        instance.school = ’RP’
        return instance
```

__str__()

```
class Human:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return ’Human[name={0}]’.format(self.name)

h = Human(’Simon’)
print(h)  # auto calls __str__()
```

Special Comparison Methods

Methods	Trigger
`__lt__`	`x < y`
`__le__`	`x <= y`
`__eq__`	`x == y`
`__ne__`	`x != y`
`__gt__`	`x > y`
`__ge__`	`x >= y`
`__hash__`	`hash(x)`
`__nonzero__`	`bool(x)`

Emulating sequence types

Methods	Trigger
`__len__`	`len(x)`
`__iter__`	`for i in x`
`__reversed__`	`reversed(x)`
`__contains__`	`i in x`

Emulating numeric types

Methods	Trigger
`__add__`	`x + 1`
`__radd__`	`1 + x`
`__iadd__`	`x += 1`
`__sub__`	`x - 1`
`__mul__`	`x * 1`
`__div__`	`x / 1`
`__pow__`	`x ** 1`
`__neg__`	`-x`

Iterators

Iterate using for loop by element

for iterates over item in the sequence [20]

```
info = ['Hello', 'World', 'Goodbye', 'Pycon']
data = []
for item in info:
    data.append(item.upper())
print('-'.join(data))
```

[20] http://docs.python.org/tutorial/controlflow.html#for-statements

Iterate using for loop by index

range() generates list containing arithmetic progressions [21]

```
info = [’Hello’, ’World’, ’Goodbye’, ’Pycon’]
length = len(info)
data = []
for index in range(length):
    item = info[index].upper()
    data.append(item)
print(’-’.join(data))
```

[21] http://docs.python.org/tutorial/controlflow.html#the-range-function

Iterate through key/value pair

- dict have the following iterators:
  - iteritems() which returns key-value pairs [22]
  - iterkeys() which returns keys only [23]
  - itervalues() which returns values only [24]

```
info = {'Hello': 'World', 'Goodbye': 'Pycon'}
data = []
for key, value in info.iteritems():
    s = 'Key {0} maps to {1}'.format(key, value)
    data.append(s)
```

[22] http://docs.python.org/library/stdtypes.html#dict.iteritems

[23] http://docs.python.org/library/stdtypes.html#dict.iterkeys

[24] http://docs.python.org/library/stdtypes.html#dict.itervalues

Iterate using Enumerate

- enumerate returns a tuple containing [25]
  - count (which starts from 0)
  - value

```
info = ['Hello', 'World', 'Goodbye', 'Pycon']
data = []
for num, item in enumerate(info):
    s = 'Num {0} is {1}'.format(num, item)
    data.append(s)
```

[25] http://docs.python.org/library/functions.html#enumerate

Non Generator function

```
def fib(n):
    a = b = 1
    result = []
    for i in range(n):
        result.append(a)
        a, b = b, a + b
    return result
```

Generator - Lazy Evaluation function

```
def fib(n):
    a = b = 1
    for i in range(n):
        yield a
        a, b = b, a + b
```

Generator from List comprehension

```
x = [1, 5, 6, 2, 8]
```

List comprehension

```
y = [item + 1 for item in x if item > 3]
```

Generator

```
z = (item + 1 for item in x if item > 3)
```

Useful Standard Library Modules for Iteration

- itertools [26]
  - Iterator functions for efficient looping (From Python 2.3)

[26] http://www.doughellmann.com/PyMOTW/itertools/index.html

Itertools Module

```
from itertools import *
from operator import itemgetter

d = dict(a=1, b=2, c=1, d=2, e=1, f=2, g=3)
di = sorted(d.iteritems(), key=itemgetter(1))
for k, g in groupby(di, key=itemgetter(1)):
    print k, map(itemgetter(0), g)
```

Exceptions

ZeroDivision, NameError and others

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name ’spam’ is not defined
>>> ’2’ + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate ’str’ and ’int’ objects
```

How to catch exceptions

```
try:
    f = open(’noexist.txt’)
    s = f.readline()
    i = int(s.strip())
except IOError as (errno, strerror):
    print(’I/O error({0}): {1}’.format(errno, strerror))
except ValueError:
    print(’Could not convert data to an integer.’)
```

Create your own exceptions

```
class MyError(Exception):
    '''
    Your exception should inherit from Exception
    '''
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return str(self.value)
```

How to raise exceptions

```
try:
    raise MyError(2 * 2)
except MyError as e:
    print(’My exception occurred, value:’ + str(e.value))
```

Cleaning up after exceptions

```
>>> def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print(’division by zero!’)
...     finally:
...         print(’executing finally clause’)
...
>>> divide(2, 1)
result is 2
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
```

Using try/except to catch uninstalled libraries

```
try:
    from cStringIO import StringIO
    # Faster C implementation
except ImportError:
    from StringIO import StringIO
    # Portable Python implementation
```

Testing

Using assert

```
>>> x = -3
>>> assert x > 0

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    assert x > 0
AssertionError
```

Using doctest

```
def square(x):
    '''
    >>> square(2)
    4
    >>> square(3)
    9
    '''
    return x * 2

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Using unittest

```
import unittest


def square(x):
    return x * x


class SquareTest(unittest.TestCase):
    def test_positive(self):
        self.assertEqual(4, square(2))

    def test_negative(self):
        self.assertEqual(4, square(-2))
```

Useful testing resources

- nose [27]
    - Nose extends unittest to make testing easier
- pyhamcrest [28]
    - Hamcrest framework for matcher objects

[27] http://readthedocs.org/docs/nose/en/latest/

[28] http://pypi.python.org/pypi/PyHamcrest

Thank you