PyCon 2009 - A Tour of Python Standard Library Documentation

Release 2

Senthil Kumaran

March 19, 2009

CONTENTS

Contents:

CHAPTER

ONE

A TOUR OF PYTHON STANDARD LIBRARY

Conference PyCon 2009

Presentor O.R.Senthil Kumaran <orsenthil@gmail.com>

CHAPTER

TWO

SOMETHING ABOUT PYTHON

2.1 Have you watched Ratatouille?

- Anyone can cook. ~ Gusteau
- Computer Programming for Everybody. ~ Guido.

2.2 What is Python?

- Middle-layer between shell and system
- Easy to use for end programmers.
- Also convenient for library programmers.
- Multiple implementations: CPython, Jython, IronPython, PyPy
- Designed by Implementation, but still a very much designed languaged.
- Feature releases happen every 18 months and bug fix releases once in 3 months.
- Active Developer and User community.
- Python 2.x and Python 3k

2.3 Python Standard Library

Python's standard library is very extensive and it offers a wide range of facilities. The library contains built-in modules (written in C) that provide access to system functionality and also modules written in Python that provide standardized solutions for many problems that occur in everyday programming.

Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

Many large projects, both student level projects and industrial projects can be quickly accomplished by effective usage of the Python Standard Library modules.

Certain modules, which are written in C, are built into the interpreter. You can find the builtin modules in the following way:

```
>>> import sys
>>> print sys.builtin_module_names
```

CHAPTER

THREE

DIVING IN

Lets dissect a working Python Program and try to understand various aspects of it.

```
import socket
SERVER = 'irc.freenode.net'
PORT = 6667
NICKNAME = 'phoe6'  # REPLACE WITH YOUR USERNAME
CHANNEL = '#python' # CHANGE CHANNEL IF DESIRED

IRC = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

def irc_conn():
    IRC.connect((SERVER,PORT))

def send_data(command):
    IRC.send(command + '\r\n')

def join(channel):
    send_data("JOIN %s" % channel)

def login(nickname, username='phoe6', password=None, realname='Senthil',
          hostname='freenode', servername='Server'):
    import getpass
    password = getpass.getpass('Enter password for %s:' % nickname)
    send_data("PASS %s" % password)
    send_data("USER %s %s %s %s" % (username, hostname, servername, realname))
    send_data("NICK %s" % nickname)

def part():
    send_data("PART")

irc_conn()
login(NICKNAME)
join(CHANNEL)
try:
    while True:
        buffer = IRC.recv(1024)
        msg = buffer.split()
        if msg[0] == "PING":
            # answer PING with PONG, as RFC 1459 specifies
            send_data("PONG %s" % msg[1])
        if msg[1] == 'PRIVMSG':
            nick_name = msg[0][:msg[0].find("!")]
            message = ' '.join(msg[3:])
```

```
            print nick_name.lstrip(':'), '->', message.lstrip(':')
finally:
    part()
```

In the above program, you have

- modules
- function definitions
- try and finally block
- socket calls.
- print statement

CHAPTER

FOUR

PY3K, WHAT TO LOOK FOR IN BRIEF

- Simpler built-in types. Instead of having a built-in type for int and then for long as in Py26, have a single built-in type int in py3k, which will behave like long and serve for int as well.
- In py26, you and str and unicode. Now Its just str, which is internally unicode. So all strings are unicode in py3k. There is a separate bytes type for bytes of characters.
- 1/5 will be 0.2 in Python3k. If you want the result to be 0, like in Python26, do 1//5
- No comparisons supported between incompatible types. Documents from time immemorial advised the users to not to rely and it can change anytime. Well the change has happened.
- Its print() function for output now, just like input() function input. Two things here. Previously in py2x, input() expected an object and raw_input() was actually used to input. Now in py3k, its just input() which will behave just like raw_input() before.
- Everything is a new style class. All classes that you define will implicitly be derived from the object class.
- There is refactoring tool developed by python hackers which can be used to port your py2x code to py3k. Everyones resounding advice is Use It!. This will help in migration as well fix any issues with the refactoring tool.

CHAPTER

FIVE

DIVING INTO PY3K

Lets look at the same program in Python 3k

```
# -*- coding: UTF-8 -*-

import socket
SERVER = 'irc.freenode.net'
PORT = 6667
NICKNAME = 'phoe6'  # REPLACE WITH YOUR USERNAME
CHANNEL = '#python' # CHANGE CHANNEL IF DESIRED

IRC = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

def irc_conn():
    IRC.connect((SERVER,PORT))

def send_data(command):
    IRC.send((command + '\r\n').encode('utf-8'))

def join(channel):
    send_data("JOIN %s" % channel)

def login(nickname, username='phoe6', password=None, realname='Senthil',
          hostname='freenode', servername='Server'):
    import getpass
    password = getpass.getpass('Enter password for %s:' % nickname)
    send_data("PASS %s" % password)
    send_data("USER %s %s %s %s" % (username, hostname, servername, realname))
    send_data("NICK %s" % nickname)

def part():
    send_data("PART")

irc_conn()
login(NICKNAME)
join(CHANNEL)
filetxt = open('irc_messages.log','a+')
try:
    while True:
        buffer = IRC.recv(1024)
        print(buffer)
        msg = buffer.split()
        if msg[0] == "PING":
            # answer PING with PONG, as RFC 1459 specifies
```

```
            send_data("PONG %s" % msg[1])
        if msg[1] == 'PRIVMSG':
            nick_name = msg[0][:msg[0].find("!")]
            message = ' '.join(msg[3:])
            filetxt.write((nick_name.lstrip(':') + '->' + message.lstrip(':') +
                           '\n').encode('utf-8'))
        filetxt.flush()
finally:
    part()
    filetxt.close()
```

- Note the change in print function.
- Sending bytes instead (encoded)

CHAPTER

SIX

OS MODULE

```
import os
import stat

filename = 'os_stat_chmod_example.txt'

if os.path.exists(filename):
    existing_permissions = stat.S_IMODE(os.stat(filename).st_mode)
    if os.access(filename, os.X_OK):
        print 'File Exists and has execute permissions.'
        print 'Removing Execute permissions'
        new_permissions = existing_permissions ^ stat.S_IXUSR
    else:
        print 'File Exists and lacks execute permission.'
        print 'Adding Execute permissions'
        new_permissions = existing_permissions | stat.S_IXUSR
else:
    print 'Creating a file and Read, Write and Execute mode'
    f = open(filename,'wt')
    f.write('contents')
    f.close()
    new_permissions = stat.S_IWRITE| stat.S_IREAD | stat.S_IXUSR

os.chmod(filename, new_permissions)


# There are several functions for working with directories on filesystem,
# including creating, listing contents and removing them.

import os
dir_name = 'os_directories_example'

print 'Creating', dir_name
os.makedirs(dir_name)

file_name = os.path.join(dir_name, 'example.txt')
print 'Creating', file_name

f = open(file_name,'wt')

try:
    f.write('example file')
finally:
    f.close()
```

```
print 'Listing:', dir_name
print os.listdir(dir_name)

print 'Cleaning up'
os.unlink(file_name)
os.rmdir(dir_name)
```

```
import os

TEST_UID = 1000  # set your user id
TEST_GID = 1000  # set your user's group id

def show_user_info():
    print('Effective User:  ', os.geteuid())
    print('Effective Group: ', os.getegid())
    print('Actual User:     ', os.getuid(),  os.getlogin())
    print('Actual Group:    ', os.getgid())
    print('Actual Groups:   ', os.getgroups())
    return


print('BEFORE CHANGE:')
show_user_info()

try:
    os.setegid(TEST_GID)
except OSError:
    print('Error: Could not change effective group. Re-run as root.')
else:
    print('CHANGED GROUP')
    show_user_info()


try:
    os.seteuid(TEST_UID)
except OSError:
    print('Error: Could not change effective user. Re-run as root.')
else:
    print('CHANGED USER')
    show_user_info()
```

```
import os

print 'Initial value:', os.environ.get('TESTVAR',None)
print 'Child Process:', os.system('echo $TESTVAR')
print


os.environ['TESTVAR'] = 'THIS VALUE HAS CHANGED.'

print 'Changed Value:', os.environ['TESTVAR']
print 'Child Process:', os.system('echo $TESTVAR')
print


del os.environ['TESTVAR']
```

```
print 'Removed Value:', os.environ.get('TESTVAR', None)
print 'Child Process:', os.system('echo $TESTVAR')


# Note the use of os.curdir and os.pardir to refer to the current and parent
# directories in a portable manner.

import os

print 'Starting:', os.getcwd()
print os.listdir(os.curdir)

print 'Moving up one:', os.pardir
os.chdir(os.pardir)

print 'After move:', os.getcwd()
print os.listdir(os.curdir)


import os

pipe_stdout = os.popen('echo "hello,world"','r')

try:
    stdout_value = pipe_stdout.read()
finally:
    pipe_stdout.close()

print stdout_value,


pipe_stdin = os.popen('cat -','w')

try:
    pipe_stdin.write('hello,world\n')
finally:
    pipe_stdin.close()


import os

pipe_stdin, pipe_stdout = os.popen2('cat -')

try:
    pipe_stdin.write('hello, world!')
finally:
    pipe_stdin.close()

try:
    stdout_value = pipe_stdout.read()
finally:
    pipe_stdout.close()

print stdout_value


import os
```

```
print 'popen2, cmd as sequence:'

pipe_stdin, pipe_stdout = os.popen2(['cat','-'])

try:
    pipe_stdin.write('through stdin to stdout')
finally:
    pipe_stdin.close()

try:
    stdout_value = pipe_stdout.read()
finally:
    pipe_stdout.close()

print stdout_value


import os

print 'popen3'

pipe_stdin, pipe_stdout, pipe_stderr = os.popen3('cat -;echo "Error Message" 1>&2')

try:
    pipe_stdin.write("Hello, World")
finally:
    pipe_stdin.close()

try:
    stdout_value = pipe_stdout.read()
finally:
    pipe_stdout.close()

try:
    stderr_value = pipe_stderr.read()
finally:
    pipe_stderr.close()

print 'STDOUT:', stdout_value
print 'STDERR:', stderr_value


import os

pipe_stdin, pipe_stdout_and_stderr = os.popen4('cat -;echo "Hello, Rats!" 1>&2')

try:
    pipe_stdin.write("Hello, World!")
finally:
    pipe_stdin.close()

try:
    stdout_value = pipe_stdout_and_stderr.read()
finally:
    pipe_stdout_and_stderr.close()

print 'Combined STDOUT and STDERR:', stdout_value
```

```
import os

print 'Testing:', __file__
print 'Exists:', os.access(__file__, os.F_OK)
print 'Readable:', os.access(__file__, os.R_OK)
print 'Writable:', os.access(__file__, os.W_OK)
print 'Executable:', os.access(__file__, os.X_OK)
```

```
import os
import sys
import time

if len(sys.argv) == 1:
    filename = __file__
else:
    filename = sys.argv[1]

stat_info = os.stat(filename)

print 'os.stat(%s):' % filename
print '\tSize:', stat_info.st_size
print '\tPermissions:', oct(stat_info.st_mode)
print '\tOwner:', stat_info.st_uid
print '\tDevice:',stat_info.st_dev
print '\tLast Modified:', time.ctime(stat_info.st_mtime)
```

```
import os, tempfile

link_name = tempfile.mktemp()

print 'Creating link %s->%s' % (link_name, __file__)
os.symlink(__file__, link_name)

stat_info = os.lstat(link_name)
print 'Permissions:', oct(stat_info.st_mode)

print 'Points to:', os.readlink(link_name)

# cleanup

os.unlink(link_name)
```

CHAPTER

SEVEN

SHUTIL MODULE

```
import os
list_of_files = os.listdir(os.getcwd())
python_files = []

for each in list_of_files:
    if each.endswith('.py'):
        python_files.append(each)

python_files.sort()
for fname in python_files:
    print fname + '\n'


from shutil import *
import os

os.mkdir('example')
print 'BEFORE:', os.listdir('example')
copy('shutil_copy.py','example')
print 'AFTER:', os.listdir('example')
os.unlink('example'+ os.sep + 'shutil_copy.py')
os.rmdir('example')


from shutil import *
import os
import time

def show_file_info(filename):
    stat_info = os.stat(filename)
    print '\tMode   :', stat_info.st_mode
    print '\tCreated :', time.ctime(stat_info.st_ctime)
    print '\tAccessed:', time.ctime(stat_info.st_atime)
    print '\tModified:', time.ctime(stat_info.st_mtime)

os.mkdir('example')
print 'SOURCE:'
show_file_info('shutil_copy2.py')
copy2('shutil_copy2.py','example')
print 'DESTINATION:'
show_file_info('example' + os.sep + 'shutil_copy2.py')
os.unlink('example' + os.sep + 'shutil_copy2.py')
os.rmdir('example')
```

```
import os

from shutil import copyfile
from glob import glob

print 'BEFORE:', glob('shutil_copyfile.*')
copyfile('shutil_copyfile.py', 'shutil_copyfile.py.copy')
print 'AFTER:', glob('shutil_copyfile.*')
os.unlink('shutil_copyfile.py.copy')
```

```
from shutil import *
import os
from StringIO import StringIO
import sys

class VerboseStringIO(StringIO):
    def read(self, n=-1):
        next = StringIO.read(self, n)
        print 'read(%d) =>' % n, next
        return next

sample_string = "Harp not on that string.-- William Shakespeare, 'Henry VI'"

print 'Default:'
input = VerboseStringIO(sample_string)
output = StringIO()
copyfileobj(input, output)

print

print 'All at Once:'
input = VerboseStringIO(sample_string)
output = StringIO()
copyfileobj(input, output, -1)

print

print 'Blocks of 20'
input = VerboseStringIO(sample_string)
output = StringIO()
copyfileobj(input, output, 20)
```

```
# By default when a new file is created under Unix, it receives permissions
# based on the umask of the current user. To copy the permissions from one file
# to another use copymode

from shutil import copymode
from commands import getstatus
import os

f = open('file_to_change.txt','wt')
f.write('content')
f.close()

os.chmod('file_to_change.txt',0444)
```

```
print 'BEFORE:', getstatus('file_to_change.txt')
copymode('shutil_copymode.py', 'file_to_change.txt')
print 'AFTER:', getstatus('file_to_change.txt')

os.unlink('file_to_change.txt')
```

```
from shutil import *
import os
import time

def show_file_info(filename):
    stat_info = os.stat(filename)
    print '\tMode   :',stat_info.st_mode
    print '\tCreated    :', time.ctime(stat_info.st_ctime)
    print '\tAccessed   :', time.ctime(stat_info.st_atime)
    print '\tModified   :', time.ctime(stat_info.st_mtime)

f = open('file_to_change.txt','wt')
f.write('content')
f.close()

os.chmod('file_to_change.txt',0444)

print 'BEFORE:'
show_file_info('file_to_change.txt')
copystat('shutil_copystat.py','file_to_change.txt')
print 'AFTER:'
show_file_info('file_to_change.txt')

os.unlink('file_to_change.txt')
```

```
import os

from shutil import copytree
from commands import getoutput

os.mkdir('example')
print 'BEFORE:'
print getoutput('ls -rlast /tmp/example')
copytree('example','/tmp/example')

print 'AFTER:'
print getoutput('ls -rlast /tmp/example')

os.rmdir('example')
os.rmdir('/tmp/example')
```

```
import os
from shutil import move
from glob import glob

f = open('example.txt','wt')
f.write('contents')
f.close()
```

```
print 'BEFORE:', glob('example*')
move('example.txt','example.out')
print 'AFTER:', glob('example*')
os.unlink('example.out')


import os

from shutil import copytree, rmtree
from commands import getoutput

os.mkdir('example')
copytree('example','/tmp/example')

print 'BEFORE:'
print getoutput('ls -rlast /tmp/example')

rmtree('/tmp/example')
print 'AFTER:'
print  getoutput('ls -rlast /tmp/example')

rmtree('example')
```

CHAPTER

EIGHT

SUBPROCESS

```
# using os.system() you could call the external operating system calls
# That is equivalent to call() method from subprocess.
# Optional shell argument provides the facility to pass the shell variables.

# Doing it os.system way
import os
os.system('date')

# Doing it subprocess way
from subprocess import call
call('date')

# Accessing shell variables
# Since we set shell=True, the shell variables are expanded in the command line
call('echo $PATH',shell=True)


# Read Output of another command

import subprocess

print '\nread:'
proc = subprocess.Popen('echo "Hello,World"',
                        shell=True,
                        stdout=subprocess.PIPE,
                       )
stdout_value = proc.communicate()[0]
print '\tstdout:',stdout_value


# Writing to the input of a pipe:

print '\nwrite:'
proc = subprocess.Popen('cat -',
                        shell=True,
                        stdin=subprocess.PIPE,
                       )
proc.communicate('\tstdin: to stdin\n')


# Reading and Writing through PIPES as with popen2

print '\npopen2:'
```

```
proc = subprocess.Popen('cat -',
                        shell=True,
                        stdin=subprocess.PIPE,
                        stdout=subprocess.PIPE,
                       )
stdout_value = proc.communicate('through stdin to stdout')[0]
print '\tpass through:',repr(stdout_value)


import subprocess
print '\npopen3:'
proc = subprocess.Popen('cat -;echo ";to stderr" 1>&2',
                        shell=True,
                        stdin=subprocess.PIPE,
                        stdout=subprocess.PIPE,
                        stderr=subprocess.PIPE,
                       )
stdout_value, stderr_value = proc.communicate('through stdin to stdout')
print '\tpass through:', repr(stdout_value)
print '\tstderr:', repr(stderr_value)
```

CHAPTER

NINE

GLOB

```
import glob

for name in glob.glob('dir/*'):
    print name


import glob

for name in glob.glob('dir/*[0-9].*'):
    print name


#!/usr/bin/env python
#
# Copyright 2007 Doug Hellmann.
#
#
#                         All Rights Reserved
#
# Permission to use, copy, modify, and distribute this software and
# its documentation for any purpose and without fee is hereby
# granted, provided that the above copyright notice appear in all
# copies and that both that copyright notice and this permission
# notice appear in supporting documentation, and that the name of Doug
# Hellmann not be used in advertising or publicity pertaining to
# distribution of the software without specific, written prior
# permission.
#
# DOUG HELLMANN DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
# INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN
# NO EVENT SHALL DOUG HELLMANN BE LIABLE FOR ANY SPECIAL, INDIRECT OR
# CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
# OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
# NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
# CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
#

"""Create test data for the glob examples.

"""

__module_id__ = "$Id: glob_maketestdata.py 1995 2009-03-01 21:09:26Z dhellmann $"
#end_pymotw_header
```

```
import os

def mkfile(filename):
    print filename
    f = open(filename, 'wt')
    try:
        f.write('\n')
    finally:
        f.close()

print 'dir'
os.mkdir('dir')

mkfile('dir/file.txt')
mkfile('dir/file1.txt')
mkfile('dir/file2.txt')
mkfile('dir/filea.txt')
mkfile('dir/fileb.txt')

print 'dir/subdir'
os.mkdir('dir/subdir')

mkfile('dir/subdir/subfile.txt')
```

```
import glob

for name in glob.glob('dir/file?.txt'):
    print name
```

```
import glob

print 'Named explicitly:'
for name in glob.glob('dir/subdir/*'):
    print '\t', name

print 'Named with wildcard'
for name in glob.glob('dir/*/*'):
    print '\t', name
```

CHAPTER

TEN

CONFIGPARSER

```
from ConfigParser import ConfigParser
import os

filename = 'config.ini'
config = ConfigParser()
config.read([filename])

url = config.get('section','url')
print url
```

CHAPTER

ELEVEN

TIME

```
import time
print 'unix time:', time.time()
print 'wallclock time:', time.ctime()


# Processor Clock time
# While time() returns a wall clock time, clock() returns processor clock time.
# The values returned from clock() should be used for performance testing,
# benchmarking etc, since they reflect the actual time used by programs and can
# be more precise than the values from time()

import hashlib
import time

# Data to use to calculate md5 checksums
data = open(__file__,'rt').read()

for i in range(5):
    h = hashlib.sha1()
    print time.ctime(), ': %0.3f %0.3f' % (time.time(), time.clock())
    for i in range(10000000):
        h.update(data)
    cksum = h.digest()


import time

for i in range(6, 1, -1):
    print '%s %0.2f %0.2f' % (time.ctime(), time.time(), time.clock())
    print 'Sleeping:', i
    time.sleep(i)


import time

now = time.ctime()
print now
parsed = time.strptime(now)
print parsed
print time.strftime("%a %b %d %H:%M:%S %Y", parsed)


import time
```

```
print 'gmtime   :', time.gmtime()
print 'localtime:', time.localtime()
print 'mktime   :', time.mktime(time.localtime())

print

t = time.localtime()

print 'Day of the month:', t.tm_mday
print 'Day of the week:', t.tm_wday
print 'Day of the year:', t.tm_yday


import time
import os

def show_zone_info():
    print '\tTZ     :', os.environ.get('TZ','(not set)')
    print '\ttzname :', time.tzname
    print '\tZone   : %d (%d)' % (time.timezone, (time.timezone / 3600))
    print '\tDST    :', time.daylight
    print '\tTime   :', time.ctime()
    print

print 'Default:'
show_zone_info()

for zone in ['US/Eastern','US/Pacific', 'GMT',
             'Europe/Amsterdam','Asia/Culcutta']:
    os.environ['TZ'] = zone
    time.tzset()
    print zone, ':'
    show_zone_info()
```

CHAPTER

TWELVE

BUILTIN MODULES

Let us start with the __builtin__ functions and exceptions which Python Standard Library defines.:

```
>>>import __builtin__
>>>dir(__builtin__)
```

12.1 all

This would list the various functions and exceptions that are available to the interpreter. Let us look into certain important ones:

```
>>> all([True,True,True,True])
True
>>> all([False,True,True,True])
False
```

12.2 any

::

```
>>> any([False,False,False,True])
True
>>> any([False,False,False,False])
False
```

12.3 callable(object)

Return True if the object argument appears callable, False if not. If this returns true, it is still possible that a call fails, but if it is false, calling object will never succeed. Note that classes are callable (calling a class returns a new instance); class instances are callable if they have a __call__() method.

12.4 staticmethod

In Object Oriented Programming, you create a method which gets associated either with a class or with an instance of the class, namely an object. This is concept is the first thing to understand.

And most often in our regular practice, we always create methods to be associated with an object. Those are called instance methods.

For e.g:

```
class Car:
        def cartype(self):
                self.model = "Audi"
mycar = Car()
mycar.cartype()
print mycar.model
```

Here cartype() is an instance method, it associates itself with an instance (mycar) of the class (Car) and that is defined by the first argument ('self').

When you want a method not to be associated with an instance, you call that as a staticmethod.

How can you do such a thing in Python?

The following would never work:

```
>>> class Car:
...     def getmodel():
...             return "Audi"
...     def type(self):
...             self.model = getmodel()
```

Because, getmodel() is defined inside the class, Python binds it to the Class Object. You cannot call it by the following way also, namely: Car.getmodel() or Car().getmodel() , because in this case we are passing it through an instance (Class Object or a Instance Object) as one of the argument while our definition does not take any argument.

As you can see, there is a conflict here and in effect the case is, It is an "unbound local method" inside the class.

Now comes Staticmethod.

Now, in order to call getmodel(), you can to change it to a static method.

::

```
>>> class Car:
...     def getmodel():
...             return "Audi"
...     getmodel = staticmethod(getmodel)
...     def cartype(self):
...             self.model = Car.getmodel()
...
>>> mycar = Car()
>>> mycar.cartype()
>>> mycar.model
'Audi'
```

Now, I have called it as Car.getmodel() even though my definition of getmodel did not take any argument. This is what staticmethod function did. getmodel() is a method which does not need an instance now, but still you do it as Car.getmodel() because getmodel() is still bound to the Class object.

12.5 Decorators

getmodel = staticmethod(getmodel)

If you look at the previous code example, the function staticmethod took a function as a argument and returned a function which we assigned to a variable (named as SAME functionname) and made it a function. Correct?

staticmethod() function thus wrapped our getmodel function with some extra features and this wrapping is called as Decorator.

The same code can be written like this.

```
>>> class Car:
...     @staticmethod
...     def getmodel():
...             return "Audi"
...     def cartype(self):
...             self.model = Car.getmodel()
...
>>> mycar = Car()
>>> mycar.cartype()
>>> mycar.model
'Audi'
```

Good reference on Decorators would be: http://personalpages.tds.net/~kent37/kk/00001.html

Please remember that this concept of Decorator is independent of staticmethod and classmethod.

12.6 classmethod

Now, what is a difference between staticmethod and classmethod?

In languages like Java,C++, both the terms denote the same :- methods for which we do not require instances. But there is a difference in Python. A class method receives the class it was called on as the first argument. This can be useful with subclasses.

We can see the above example with the classmethod and a decorator as:

```
>>>
>>> class Car:
...     @classmethod
...     def getmodel(cls):
...             return "Audi"
...     def gettype(self):
...             self.model = Car.getmodel()
...
>>> mycar = Car()
>>> mycar.gettype()
>>> mycar.model
'Audi'
```

The following are the references in order to understand further: 1) Alex-Martelli explaining it with code: http://code.activestate.com/recipes/52304/ 2) Decorators: http://personalpages.tds.net/~kent37/kk/00001.html

12.7 ellipsis

- ellipsis is another builtin. It is only used in slicing.

12.8 property

```
print "---- First Section ----"

class Name(object):
    def __init__(self):
        self.name = "PyCon 2009"
    def confname(self):
        return self.name

conference = Name()
print conference.confname
print conference.confname()


print "---- Second Section ----"

class Name2(object):
    def __init__(self):
        self.name = "PyCon 2009"

    def confname(self):
        return self.name

    confname = property(confname)

conference2 = Name2()
print conference2.confname

# print conference2.confname() is not possible, because it is an attribute and
# not callable.

print "---- Third Section ----"

class Name3(object):
    def __init__(self):
        self.name = "PyCon 2009"

    @property
    def confname(self):
        return self.name


conference3 = Name3()
print conference3.confname
```

12.9 super

```
class A(object):
    def __init__(self):
        print "A init"

class B(A):
    def __init__(self):
```

```
        print "B init"
        super(B, self).__init__()

class C(A):
    def __init__(self):
        print "C init"
        super(C, self).__init__()

class D(B,C):
    def __init__(self):
        print "D init"
        super(D, self).__init__()

obj = D()


class A(object):
    def __init__(self):
        print "A init"
        print self.__class__.__mro__

class B(A):
    def __init__(self):
        print "B init"
        print self.__class__.__mro__
        super(B, self).__init__()

class C(A):
    def __init__(self):
        print "C init"
        print self.__class__.__mro__
        super(C, self).__init__()

class D(B, C):
    def __init__(self):
        print "D init"
        print self.__class__.__mro__
        super(D, self).__init__()

x = D()


class Farm(object):
    def __init__(self):
        self.x = "I am from Farm"

class Barm(Farm):
    def __init__(self):
        super(Barm, self).__init__()

obj = Barm()
print obj.x


class Farm(object):
    def __init__(self):
        self.x = "I am from Farm"
```

```
class Barm(Farm):
    def __init__(self):
        Farm.__init__(self)

obj = Barm()
print obj.x
```

12.10 compile

```
source = "import os;print os.listdir(os.getcwd())"
obj = compile(source,'<string>','exec')
eval(obj)
exec(obj)
```

12.11 Exceptions

- Exceptions are Classes and are __builtin__ to the interpreter.
- Until 1.5, simple string messages were exceptions.
- The exception classes are defined in a hierarchy, related exceptions can be caught by catching their base classes.

12.11.1 BaseException

Baseclass for all exceptions. Implements logic for creating the string representation of the exception using the str() from the arguments passed to the constructor.

12.11.2 Exception

Baseclass for the exception that do not result in quitting the running application. All user-defined exceptions should use Exception as a base class.

12.11.3 StandardError

Baseclass for builtin exceptions used in Standard Library.

12.11.4 ArthimeticError

Baseclass for math related errors.

12.11.5 LookupError

When something cannot be found.

12.11.6 EnvironmentError

Base class for errors that come from outside of Python (the operating system, filesystem, etc.).

12.11.7 AssertionError

An AssertionError is raised by a failed assert statement.

:: >>>assert False, 'The assertion failed' >>># This should throw a simple AssertionError

12.11.8 AttributeError

When an attribute reference or assignment fails, AttributeError is raised.

::
```
>>> x = "PyCon 2009"
>>> x.imag
>>> # This would throw AttributeError
```

AttributeError will also be raised when trying to modify a read-only attribute.

```
class MyClass(object):

    @property
        def attribute(self):
                return 'This is the attribute value'

                o = MyClass()
                print o.attribute
                o.attribute = 'New value'
```

12.11.9 EOFError

An EOFError is raised when a builtin function like input() or raw_input() do not read any data before encountering the end of their input stream.

12.11.10 IOError

Raised when input or output fails, for example if a disk fills up or an input file does not exist.

:: f = open('/does/not/exist', 'r')

12.11.11 importError

Raised when a module, or member of a module, cannot be imported.

12.11.12 IndexError

An IndexError is raised when a sequence reference is out of range.

```
my_seq = [ 0, 1, 2 ]
print my_seq[3]
```

12.11.13 KeyError

A KeyError is raised when a value is not found as a key of a dictionary.

:: d = { 'a':1, 'b':2 } print d['c']

12.11.14 KeyboardInterrupt

A KeyboardInterrupt occurs whenever the user presses Ctrl-C (or Delete) to stop a running program. Unlike most of the other exceptions, KeyboardInterrupt inherits directly from BaseException to avoid being caught by global exception handlers that catch Exception.

```
try:
    print 'Press Return or Ctrl-C:',
    ignored = raw_input()
except Exception, err:
    print 'Caught exception:', err
except KeyboardInterrupt, err:
    print 'Caught KeyboardInterrupt'
else:
    print 'No exception'
```

12.11.15 MemoryError

If your program runs out of memory and it is possible to recover (by deleting some objects, for example), a MemoryError is raised.

```
import itertools

# Try to create a MemoryError by allocating a lot of memory
l = []
for i in range(3):
    try:
        for j in itertools.count(1):
            print i, j
            l.append('*' * (2**30))
    except MemoryError:
        print '(error, discarding existing list)'
        l = []
```

12.11.16 NameError

NameErrors are raised when your code refers to a name that does not exist in the current scope. For example, an unqualified variable name.

12.11.17 NotImplementedError

User-defined base classes can raise NotImplementedError to indicate that a method or behavior needs to be defined by a subclass, simulating an interface.

CHAPTER

THIRTEEN

URLLIB

```
import urllib

query_args = {'q':'query_string', 'foo':'bar'}
encoded_args = urllib.urlencode(query_args)
print 'Encoded:', encoded_args

url = 'http://localhost:8000/?' + encoded_args
print urllib.urlopen(url).read()
```

```
import urllib

response = urllib.urlopen('http://www.example.com')

for line in response:
    print line.rstrip()
```

```
import urllib

response = urllib.urlopen('http://localhost:8080/')
print 'RESPONSE:', response
print 'URL      :', response.geturl()

headers = response.info()

print 'DATE:', headers['date']
print 'HEADERS:'
print '---------'
print headers

data = response.read()
print 'LENGTH   :', len(data)
print 'DATA     :'
print '-----------'
print data
```

```
import os
from urllib import pathname2url, url2pathname

print '==Default=='

path = '/a/b/c'
```

```
print 'Original:', path
print 'URL:', pathname2url(path)
print 'Path:', url2pathname('/d/e/f')


from nturl2path import pathname2url, url2pathname

print '== Windows without a Drive Letter =='
path = path.replace('/','\\')
print 'Original:', path
print 'URL:', pathname2url(path)
print 'Path:', url2pathname('/d/e/f')

print '== Windows URL with a Drive Letter =='
path = 'C:\\' + path.replace('/','\\')

print 'Original:', path
print 'URL     :', pathname2url(path)
print 'Path    :', url2pathname('/d/e/f')


import urllib

query_args = {'q':'query string', 'foo':'bar'}
encoded_args = urllib.urlencode(query_args)
url = 'http://localhost:8000/'
print urllib.urlopen(url, encoded_args).read()


import urllib

url = 'http://uthcode.sarovar.org/~senthil'
print 'urlencode():', urllib.urlencode({'url':url})
print 'quote():', urllib.quote(url)
print 'quote_plus():', urllib.quote_plus(url)

quoted = urllib.quote(url)
quoteplused = urllib.quote_plus(url)

print 'unquote', urllib.unquote(quoted)
print 'unquote_plus', urllib.unquote_plus(quoteplused)


import urllib
import os

def reporthook(blocks_read, block_size, total_size):
    if not blocks_read:
        print 'Connection opened'
        return
    if total_size < 0:
        # unknown size
        print 'Read %d blocks' % blocks_read
    else:
        amount_read = blocks_read * block_size
        print 'Read %d blocks, or %d %d' % (blocks_read, amount_read,
                                            total_size)
        return
```

```
try:
    filename, msg = urllib.urlretrieve('http://www.example.com',
                                       reporthook=reporthook)
    print
    print 'File:', filename
    print 'Headers:'
    print msg
    print 'File exists before cleanup:', os.path.exists(filename)

finally:
    urllib.urlcleanup()

    print 'File still exists:', os.path.exists(filename)
```

CHAPTER

FOURTEEN

BASEHTTPSEVER

```
import BaseHTTPServer
import cgi, random, sys

MESSAGES = [
    "I like maxims that don't encourage behavior modification.",
    "Reality continues to ruin my life.",
    "Weekends don't count unless you spend them doing something completely \
    pointless.",
    "Life's disappointments are harder to take when you don't know any swear \
    words."
]

class Handler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path != "/":
            self.send_error(404,"File not Found!")
            return
        self.send_response(200)
        self.send_header("Content-type:", "text/html")
        self.end_headers()
        try:
            # redirect stdout to client
            stdout = sys.stdout
            sys.stdout = self.wfile
            self.makepage()
        finally:
            sys.stdout = stdout # restore

    def makepage(self):
        # generate a random message
        tagline = random.choice(MESSAGES)
        print "<html>"
        print "<body>"
        print "<p>Today's Quote:"
        print "<i>%s</i>" % cgi.escape(tagline)
        print "</body>"
        print "</html>"

PORT = 8000

httpd = BaseHTTPServer.HTTPServer(("",PORT), Handler)
print "serving at port", PORT
httpd.serve_forever()
```

CHAPTER

FIFTEEN

SIMPLEXMLPRCSERVER

```
from SimpleXMLRPCServer import SimpleXMLRPCServer
import logging
import os

# Set up logging
logging.basicConfig(level=logging.DEBUG)

server = SimpleXMLRPCServer(('localhost',9000), logRequests=True)

# Expose a function
def list_contents(dir_name):
    logging.debug('list_contents(%s)', dir_name)
    return os.listdir(dir_name)

server.register_function(list_contents)

try:
    print 'Use Control-C to quit.'
    server.serve_forever()
except KeyboardInterrupt:
    print 'Exiting'
```

```
from SimpleXMLRPCServer import SimpleXMLRPCServer
import os

server = SimpleXMLRPCServer(('localhost',9000), allow_none=True)

server.register_function(os.listdir, 'dir.list')
server.register_function(os.mkdir,'dir.create')
server.register_function(os.rmdir,'dir.remove')

try:
    print 'Use Control-C to quit'
    server.serve_forever()
except KeyboardInterrupt:
    print 'Exiting'
```

```
from SimpleXMLRPCServer import SimpleXMLRPCServer, list_public_methods
import os
import inspect

server = SimpleXMLRPCServer(('localhost', 9000), logRequests=True)
```

```
server.register_introspection_functions()

class DirectoryService:

    def _listMethods(self):
        return list_public_methods(self)

    def _methodHelp(self, method):
        f = getattr(self, method)
        return inspect.getdoc(f)

    def list(self, dir_name):
        """list(dir_name) => [<filename>]

        Returns a list containing the contents of the named directory.
        """
        return os.listdir(dir_name)


server.register_instance(DirectoryService())

try:
    print 'Use CTRL-C to exit'
    server.serve_forever()
except KeyboardInterrupt:
    print 'Exitting'
```

15.1 xmlrpclib

```
import xmlrpclib

proxy = xmlrpclib.ServerProxy('http://localhost:9000')
print proxy.list_contents('/home/senthil')
```

```
import xmlrpclib

proxy = xmlrpclib.ServerProxy('http://localhost:9000')
print 'BEFORE   :', 'Example' in proxy.dir.list('/home/senthil')
print 'CREATE   :', proxy.dir.create('/home/senthil/Example')
print 'SHOULD EXIST :', 'Example' in proxy.dir.list('/home/senthil/')
print 'REMOVE   :', proxy.dir.remove('/home/senthil/Example')
print 'AFTER    :', 'Example' in proxy.dir.list('/home/senthil')
```

```
import xmlrpclib

proxy = xmlrpclib.ServerProxy('http://localhost:9000')

for method_name in proxy.system.listMethods():
    print '=' * 60
    print method_name
    print '-' * 60
    print proxy.system.methodHelp(method_name)
    print
```

CHAPTER

SIXTEEN

SMTPD AND SMTPLIB

```
import smtpd
import asyncore

class CustomSMTPServer(smtpd.SMTPServer):

    def process_message(self, peer, mailfrom, rcpttos, data):
        print 'Receiving message from:', peer
        print 'Message addressed from:', mailfrom
        print 'Message addressed to  :', rcpttos
        print 'Message length        :', len(data)
        print 'Message is            :', data
        return

server = CustomSMTPServer(('127.0.0.1',1025), None)
asyncore.loop()


import smtplib
import email.utils
from email.mime.text import MIMEText

# Create the message

msg = MIMEText('Well, it just seemed wrong to cheat on an ethics test.  --\
               Calvin')
msg['To'] = email.utils.formataddr(('Recipient','recipient@example.com'))
msg['From'] = email.utils.formataddr(('Author','author@example.com'))
msg['Subject'] = 'I am home, Hobbes.'

server = smtplib.SMTP('127.0.0.1', 1025)
server.set_debuglevel(True) # Logs the communication with the server

try:
    server.sendmail('author@example.com',
                    ['recipient@example.com'],msg.as_string())
finally:
    server.quit()
```

CHAPTER

SEVENTEEN

SOCKET MODULE

The Python socket module provides direct access to the standard BSD socket interface, which is available on most modern computer systems. The advantage of using Python for socket programming is that socket addressing is simpler and much of the buffer allocation is done automatically. In addition, it is easy to create secure sockets and several higher-level socket abstractions are available.

To create a server, you need to:

1. create a socket
2. bind the socket to an address and port
3. listen for incoming connections
4. wait for clients
5. accept a client
6. send and receive data

To create a client, you need to:

1. create a socket
2. connect to the server
3. send and receive data

```
import socket

host = '127.0.0.1'
port = 50000
size = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
s.send("Hello,World")
data = s.recv(size)
s.close()
print 'Received:', data
```

```
import socket

host = '127.0.0.1'
```

```
port = 50000
backlog = 5
size = 1024
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((host, port))
s.listen(backlog)
while True:
    client, address = s.accept()
    data = client.recv(size)
    if data:
        client.send(data)
    client.close()
```

17.1 Additional Notes of Socket Programming.

17.1.1 Addresses

The BSD socket interface defines several different types of addresses called families. These include:

- AF_UNIX: A Unix socket allows two processes running on the same machine to communicate with each other through the socket interface. In Python, UNIX socket addresses are represented as a string.
- AF_INET: An IPv4 socket is a socket between two processes, potentially running on different machines, using the current version of IP (IP version 4). This is the type of socket most programs use today. In Python, IPv4 socket addresses are represented using a tuple of (host, port), where host is a string host name and port is an integer called the port number. For the host name you can specify either a standard Internet name, such as 'www.cnn.com' or an IP address in dotted decimal notation, such as '64.236.24.20'.
- AF_INET6: An IPv6 socket is similar to an IPv4 socket, except that it uses IPv6. The main change in IPv6 is that it uses 128 bit addresses, whereas IPv4 uses only 32 bits; this allows IPv6 to better meet the current high demand for IP

  addresses. In addition, IPv6 uses flow identifiers to provide different Quality of Service to applications (i.e. low delay or guaranteed bandwidth) and scope identifiers to limit packet delivery to various administrative boundaries. In Python, IPv6 socket addresses are represented using a tuple of (host, port, flowinfo, scopeid), where flowinfo is the flow identifier and scopeid is the scope identifier. Since support of IPv6 in many host operating systems is still incomplete, we will not discuss IPv6 further in this tutorial.

17.1.2 Sockets

To create a socket in Python, use the socket() method

```
socket(family,type[,protocol])
```

The family is either AF_UNIX, AF_INET, or AF_INET6. There are several different types of sockets; the main ones are SOCK_STREAM for TCP sockets and SOCK_DGRAM for UDP sockets. You can skip the protocol number in most cases.

Please note that the constants listed above for socket family and socket type are defined inside the socket module. This means that you must import the socket module and then reference them as 'socket.AF_INET'.

The important thing about the socket() method is it returns a socket object. You can then use the socket object to call each of its methods, such as bind, listen, accept, and connect.

17.1.3 s.listen(backlog)

This tells the operating system to keep a backlog of five connections. This means that you can have at most five clients waiting while the server is handling the current client. You can set this higher, but the operating system will typically allow a maximum of 5 waiting connections. To cope with this, busy servers need to generate a new thread to handle each incoming connection so that it can quickly serve the queue of waiting clients.

http://ilab.cs.byu.edu/python/socketmodule.html

CHAPTER

EIGHTEEN

THREADING

```
import threading, zipfile

class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        threading.Thread.__init__(self)
        self.infile = infile
        self.outfile = outfile

    def run(self):
        f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_DEFLATED)
        f.write(self.infile)
        f.close()
        print 'Finished background zip of : ', self.infile

background = AsyncZip('mydata.txt','myarchive.zip')
background.start()
print 'The main program continues to run in the foreground.'

background.join()
print 'Main program waited until background was done.'
```

```
import threading

class MyThread(threading.Thread):

    def run(self):
        print 'Insert some thread stuff here.'
        print 'It will be executed.'
        print 'There is nothing much here.'

# To create a thread, just call its start() method.
# That is it.

MyThread().start()
```

```
from threading import Thread

class MyThread(Thread):
    def run(self):
        print "I am a thread!"

foobar = MyThread()
```

```
foobar.start()
foobar.join()


from threading import Thread

class myObject(Thread):
    def __init__(self):
        self._val = 1
    def get(self):
        return self._val
    def increment(self):
        self._val += 1

def t1(ob):
    ob.increment()
    print 't1:', ob.get() == 2

def t2(ob):
    ob.increment()
    print 't2:', ob.get() == 2

ob = myObject()

# Create two threads modifying the same ob instance

thread1 = Thread(target=t1, args=(ob,))
thread2 = Thread(target=t2, args=(ob,))


# Run the thread

thread1.start()
thread2.start()
thread1.join()
thread2.join()


# Instead of a single thread, create a group of threads.

import threading

theVar = 1

class MyThread(threading.Thread):

    def run(self):
        global theVar
        print 'This is a thread' + str(theVar) + 'speaking'
        print 'Hello and Good Bye'
        theVar = theVar + 1


for x in xrange(20):
    MyThread().start()
```

```
import pickle
import socket
import threading

# We will pickle a list of numbers

somelist = [1, 2, 7, 9, 0]
pickledlist = pickle.dumps(somelist)


# Our thread class

class ClientThread(threading.Thread):

    # Override Thread's __init__ method to accept the parameters needed.
    def __init__(self, channel, details):
        self.channel = channel
        self.details = details
        threading.Thread.__init__(self)

    def run(self):
        print 'Received Connection:', self.details[0]
        self.channel.send(pickledlist)
        for x in xrange(10):
            print self.channel.recv(1024)
        self.channel.close()
        print 'Closed Connection:', self.details[0]


# Setup the server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('',2727))
server.listen(5)

# Have the server serve forever
while True:
    channel, details = server.accept()
    ClientThread(channel, details).start()


import pickle
import socket


# Connect to the Server

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((('localhost'), 2727))

print pickle.loads(client.recv(1024))


# Send some messages

for x in xrange(10):
    client.send('Hey,' + str(x) + '\n')

# Close the connection
```

```
client.close()


import threading
import time

class TestThread(threading.Thread):
    def run(self):
        print 'Patient: Doctor, am I going to die?'

class AnotherThread(TestThread):
    def run(self):
        TestThread.run(self)
        time.sleep(10)

dying = TestThread()
dying.start()

if dying.isAlive():
    print 'Doctor, No'
else:
    print 'Oops. Dead before I could do anything. Next!'

living = AnotherThread()
living.start()

if living.isAlive():
    print 'Doctor:No'
else:
    print 'Doctor: Next!'


from threading import Thread
from operator import add
import random

class Bank(object):
    def __init__(self, naccounts, ibalance):
        self._naccounts = naccounts
        self._ibalance = ibalance

        self.accounts = []
        for n in range(self._naccounts):
            self.accounts.append(self._ibalance)

    def size(self):
        return len(self.accounts)

    def getTotalBalance(self):
        return reduce(add, self.accounts)

    def transfer(self, name, afrom, ato, amount):

        if self.accounts[afrom] < amount: return

        self.accounts[afrom] -= amount
        self.accounts[ato] += amount
```

```
        print "%-9s %8.2f from %2d to %2d Balance: %10.2f" % \
                (name, amount, afrom, ato, self.getTotalBalance())


class transfer(Thread):
    def __init__(self, bank, afrom, maxamt):
        Thread.__init__(self)
        self._bank = bank
        self._afrom = afrom
        self._maxamt = maxamt

    def run(self):
        while True:
        #for i in range(0, 3000):
            ato = random.choice(range(b.size()))
            amount = round((self._maxamt * random.random()), 2)
            self._bank.transfer(self.getName(), self._afrom, ato, amount)

naccounts = 2
initial_balance = 10

b = Bank(naccounts, initial_balance)

threads = []

for i in range(0, naccounts):
    threads.append(transfer(b, i, 10))
    threads[i].start()

for i in range(0, naccounts):
    threads[i].join()


import pickle
import socket
import threading

# Here's our thread

class ConnectionThread(threading.Thread):
    def run(self):
        # Connect to the server
        client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        client.connect(('localhost',2727))

        # Retrive and unpickle the list object
        print pickle.loads(client.recv(1024))

        # Send some messages
        for x in xrange(10):
            client.send('Hey,' + str(x) + '\n')

        # Close the connection
        client.close()


# Lets spawn few threads
```

```
for x in xrange(5):
    ConnectionThread().start()


import thread

def athread(stuff):
    print "I' am a real boy!"
    print stuff

thread.start_new_thread(athread, ('Argument',))


import threading
import time


class ThreadOne(threading.Thread):
    def run(self):
        print 'Thread', self.getName(), 'started.'
        time.sleep(5)
        print 'Thread', self.getName(), 'ended.'


class ThreadTwo(threading.Thread):
    def run(self):
        print 'Thread', self.getName(), 'started.'
        thingOne.join()
        print 'Thread', self.getName(), 'ended.'

thingOne = ThreadOne()
thingOne.start()
thingTwo = ThreadTwo()
thingTwo.start()


import threading

class TestThread(threading.Thread):
    def run(self):
        print 'Hello, my name is', self.getName()


bombay = TestThread()
bombay.setName('bombay')
bombay.start()


madras = TestThread()
madras.setName('madras')
madras.start()


TestThread().start()
TestThread().start()
bangalore = TestThread()
bangalore.setName('bangalore')
```

```
bangalore.start()
TestThread().start()


import pickle
import Queue
import socket
import threading


# We'll pickle a list of numbers, yet again:

somelist = [1,2,7,9,0]
pickledlist = pickle.dumps(somelist)

# A revised version of the thread class

class ClientThread(threading.Thread):
    # Note that we do not override Thread's __init__ method
    # The Queue module makes this as not neccessary

    def run(self):
        # Have our thread serve "forever".
        while True:
            # Get a client out of the queue.
            client = clientPool.get()

            # Check if actually have an actual client in the client variable.

            if client != None:
                print 'Received Connection:', client[1][0]
                client[0].send(pickledlist)
                for x in xrange(10):
                    print client[0].recv(1024)

                client[0].close()
                print 'Closed Connection:', client[1][0]



# Create our Queue

clientPool = Queue.Queue(0)

# Start two threads
for x in xrange(2):
    ClientThread().start()

# Setup the server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('', 2727))
server.listen(5)

# Have the server 'serve' forever

while True:
    clientPool.put(server.accept())
```

```
import threading
import time

class DaemonThread(threading.Thread):
    def run(self):
        self.setDaemon(True)
        time.sleep(10)

DaemonThread().start()
print 'Leaving.'
```

```
from threading import Thread
import time

def myfunc():
    print "hello, world"
    time.sleep(4)

thread1 = Thread(target=myfunc)
thread1.start()
thread1.join(timeout=5)
print thread1.isAlive()
```

CHAPTER

NINETEEN

LOGGING

```
import logging

LOG_FILENAME = 'logging_example.out'

logging.basicConfig(filename=LOG_FILENAME,
                    level=logging.DEBUG,
                   )
logging.debug('This message should go to the log file')

f = open(LOG_FILENAME, 'rt')

try:
    body = f.read()
finally:
    f.close()

print 'FILE:', LOG_FILENAME
print body


import logging
import sys

LEVELS = {'debug': logging.DEBUG,
          'info': logging.INFO,
          'warning': logging.WARNING,
          'error':logging.ERROR,
          'critical':logging.CRITICAL,
         }

if len(sys.argv) > 1:
    level_name = sys.argv[1]
    level = LEVELS.get(level_name, logging.NOTSET)
    logging.basicConfig(level=level)

logging.debug('This is a debug message')
logging.info('This is an informational message')
logging.warning('This is an warning message')
logging.error('This is an error message.')
logging.critical('This is critical error message')


import logging
```

```
logging.basicConfig(level=logging.WARNING)

logger1 = logging.getLogger('package1.module1')
logger2 = logging.getLogger('package2.module2')

logger1.warning('This message comes from one module')
logger2.warning('This message comes from another module')
```

```
import glob
import logging
import logging.handlers

LOG_FILENAME = 'logging_rotatingfile_example.out'

# Set up a specific logger with our desired output level
my_logger = logging.getLogger('MyLogger')
my_logger.setLevel(logging.DEBUG)

# Add the log message handler to the logger
handler = logging.handlers.RotatingFileHandler(LOG_FILENAME, maxBytes=20,
                                               backupCount=5)
my_logger.addHandler(handler)

# Log some messages

for i in range(20):
    my_logger.debug('i= %d' % i)

# See what files are created

logfiles = glob.glob('%s*' % LOG_FILENAME)

for filename in logfiles:
    print filename
```

CHAPTER

TWENTY

DOCTEST

```
def add(a, b):
    """Add two arbitrary objects and return their sum.
    >>> add(1, 2)
    3
    >>> add([1],[2])
    [1, 2]
    >>> add('Calvin ','& Hobbes')
    'Calvin & Hobbes'
    >>>
    >>> add([1], 2)
    Traceback (most recent call last):
    TypeError: can only concatenate list (not "int") to list
    """
    return a + b

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


def add(a,b):
    return a+b

if __name__ == '__main__':
    import unittest, doctest
    suite = doctest.DocFileSuite('doctest_tests.txt')
    unittest.TextTestRunner().run(suite)
```

CHAPTER

TWENTYONE

UNITTEST

```
#!/usr/bin/python

# This is for different methods to do unittests.
# Should serve as a handy reference and quick-lookup instead of digging into the
# docs.

import unittest

def raises_error(*args, **kwargs):
    print args, kwargs
    raise ValueError("Invalid Value:" + str(args), str(kwargs))

class UnitTestExamples(unittest.TestCase):

    def setUp(self):
        print 'In setUp()'
        self.fixture = range(1,10)

    def tearDown(self):
        print 'In tearDown()'
        del self.fixture

    # The following 3 Testcases are Same.

    def testAssertAlmostEqual(self):
        self.assertAlmostEqual(3.1, 4.24-1.1, places=1,msg="Upto 1 decimal place.")

    def testAssertAlmostEquals(self):
        self.assertAlmostEquals(3.1, 4.24-1.1, places=1,msg="Upto 1 decimal place.")

    def testFailUnlessAlmostEqual(self):
        self.failUnlessAlmostEqual(3.1, 4.24-1.1, places=1,msg="Upto 1 decimal place.")

    # The following 3 Testcases are same.

    def testAssertEqual(self):
        self.assertEqual(True,True)
        self.assertEqual(False,False)

    def testAssertEquals(self):
        self.assertEqual(True,True)
        self.assertEquals(False,False)
```

```
    def testFailUnlessEqual(self):
        self.failUnlessEqual(True,True)
        self.assertEquals(False,False)

    # The following 2 testcases are same.

    def testAssertFalse(self):
        self.assertFalse(False)

    def testFailIf(self):
        self.failIf(False)

    # The following 3 testcases are same.

    def testAssertNotAlmostEqual(self):
        self.assertNotAlmostEqual(1.1,1.9,places=1)

    def testAssertNotAlmostEquals(self):
        self.assertNotAlmostEqual(1.1,1.9,places=1)

    def testFailIfAlmostEqual(self):
        self.failIfAlmostEqual(1.1,1.9,places=1)

    # The following 3 testcases are same.

    def testAssertNoEqual(self):
        self.assertNotEqual(True,False)
        self.assertNotEqual(False,True)

    def testAssertNoEquals(self):
        self.assertNotEquals(True,False)
        self.assertNotEquals(False,True)

    def testFailIfEqual(self):
        self.failIfEqual(False,True)

    # The following 2 testcases are same.

    def testAssertRaises(self):
        self.assertRaises(ValueError, raises_error, 'a',b='c')

    def testFailUnlessRaises(self):
        self.failUnlessRaises(ValueError, raises_error, 'a',b='c')

    # The following 3 testcases are same.

    def testAssertTrue(self):
        self.assertTrue(True)

    def testFailUnless(self):
        self.failUnless(True)

    def testassert_(self):
        self.assert_(True)

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

CHAPTER

TWENTYTWO

PYTHON SHORTCUTS

```
import copy
new_list = copy.copy(existing_list)

# When we want every item and attribute in the object to be separately copied,
# recursively use deepcopy

new_list_of_dicts = copy.deepcopy(existing_list_of_dicts)

# For normal shallow copies, we have a good alternative to copy.copy, if we know
# the type of object we want to copy. To copy a list L, call list(L); to copy a
# dict d, call dict(d), to copy a set call set(s).

# To copy a a copyable object o, which belongs to some built-in Python type t,
# we may generally just call t(o)


# The way to shorten a nested list.

def list_or_tuple(x):
    return isinstance(x, (list, tuple))

def flatten(sequence, to_expand=list_or_tuple):
    for item in sequence:
        if to_expand(item):
            for subitem in flatten(item, to_expand):
                yield subitem
        else:
            yield item

nested_list = [1,2,[3,4,5],6,7,[8,[9,[10],11,12],13]]

for x in flatten(nested_list):
    print x,


import os
import sys
import urllib

def _reporthook(numblocks, blocksize, filesize, url=None):
    # print "_reporthook(%s, %s, %s)" % (numblocks, blocksize, filesize)
    base = os.path.basename(url)
    #XXX Should handle the possible filesize= -1
    try:
```

```
        percent = min((numblocks*blocksize*100)/filesize, 100)
    except:
        percent = 100
    if numblocks != 0:
        sys.stdout.write("\b"*70)
    sys.stdout.write("%-66s%3d%%" % (base, percent))

def geturl(url, dst):
    print "get url '%s' to '%s'" % (url, dst)
    if sys.stdout.isatty():
        urllib.urlretrieve(url, dst,
                           lambda nb, bs, fs, url=url:_reporthook(nb, bs, fs,
                                                                  url))
        sys.stdout.write('\n')
    else:
        urllib.urlretrieve(url, dst)

if __name__ == '__main__':
    if len(sys.argv) == 2:
        url = sys.argv[1]
        base = url[url.rindex('/')+1:]
        geturl(url, base)
    elif len(sys.argv) == 3:
        url, base = sys.argv[1:]
        geturl(url, base)
    else:
        print "Usage: geturl.py URL [DEST]"
        sys.exit(1)


import string

for n in dir(string):
    if n.startswith('_'):
        continue
    v = getattr(string, n)
    if isinstance(v, basestring):
        print '%s %s' % (n, repr(v))
        print
```

CHAPTER

TWENTYTHREE

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*