
CS 696 - Introduction to Grid Computing:

Lecture #4

Mary Thomas
San Diego State

Tuesday, 30-Jan-07

Basics

- Mailing List: please sign up
 - <http://scilists.sdsu.edu/mailman/listinfo.cgi/cs696-grid>
- References:
 - Python Tutorial & Docs, Guido van Rossum,
 - <http://www.python.org/doc/current/tut/tut.html>
 - Learning Python, Lutz & Ascher, O'Reilly, 1999
 - Web Programming in Python, by Thiruvathukal, et al.
 - Out of print, but we will get electronic copy soon.

UTILITIES

Running Python

Interpreter

Script file

Compiled code

Application

Interpreter

```
AI 11->python
```

```
Python 2.3.5 (#1, Mar 20 2005, 20:38:20)
```

```
Type "help" for more information.
```

```
>>> 1 + 2
```

```
3
```

```
>>>
```

Running Using Script File

Unix Example

File name: helloWorld.py

```
#!/usr/bin/env pythonprint 'Hello  
World'
```

Make the file executable:

```
AI 45->chmod u+x helloWorld.py
```

Run the file

```
AI 46->helloWorld.py
```

Importing modules

```
[gidget:% chmod 755 fibo.py  
[gidget:% ls fibo.py  
-rwxr-xr-x  ... fibo.py
```

```
#Fibonacci numbers module  
#return Fibonacci series up to n  
  
def fib(n):  
    a, b = 0, 1  
    while b < n :  
        print b,  
        a, b = b, a+b  
  
def fib2(n):  
    result = []  
    Print 'fib2'  
    a, b = 0, 1  
    while b < n :  
        result.append(b)  
        a, b = b, a+b  
    return result
```

```
>>> import fibo  
>>> fibo.fib(20)  
1 1 2 3 5 8 13  
>>> fibo.fib2(50)  
[1, 1, 2, 3, 5, 8, 13, 21, 34]  
  
# edit changes fibo.py  
>>> reload(fibo)  
>>> fibo.fib2(50)  
fib2  
[1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Creating PyDocs

file: fibo.py

"Fibonacci numbers module"

```
def fib(n):
    a, b = 0, 1
    while b < n :
        print b,
        a, b = b, a+b

#return Fibonacci series up to n
def fib2(n):
    "this is fib2"
    result = []
    print "fib2"
    a, b = 0, 1
    while b < n :
        result.append(b)
        a, b = b, a+b
    return result
```

Help on module fibo:

NAME

fibo - Fibonacci numbers module

FILE

/Users/mthomas/working/python/cs696/fibo.py

FUNCTIONS

fib(n)

fib2(n)

this is fib2

File I/O

File modes: 'r', 'w', 'a' (append)

```
>>> sampleFile = open('test', 'w')
>>> sampleFile.write('This is a test\n')
>>> shoppingList = 'bread' , 'cheese', 'lentils'
>>> shoppingList
('bread', 'cheese', 'lentils')
>>> sampleFile.writelines(shoppingList)
>>> sampleFile.writelines(shoppingList)
>>> sampleFile.close()
# or use sampleFile.flush()
>>> sampleFile = open('test', 'r')
>>> sampleFile.readline()
'This is a test\n'
>>> sampleFile.readline()
'breadcheeselentilsbreadcheeselentils'

>>> sampleFile = open('test', 'r')
>>> sampleFile.readline()
'This is a test\n'
>>> sampleFile.readlines()
['breadcheeselentilsbreadcheeselentils']
```

Details

<http://docs.python.org/lib/bltin-file-objects.html>

Be careful of default write **location**.

OS - navigation

```
>>> import os
>>> dir=os.getcwd()
>>> dir
'/Users/mthomas/working/python/cs696'
>>> os.chdir('/Users/mthomas/working/python')
>>> os.getcwd()
'/Users/mthomas/working/python'
>>> os.chdir('..')
>>> os.getcwd()
'/Users/mthomas/working'

>>> os.listdir('.')
['.findorders.py.swp', '.tstsum.py.swp', ... ]
>>> os.listdir(os.getcwd())
['.findorders.py.swp', '.tstsum.py.swp', ... ]
>>>
```

Modules and File I/O

```
f=open('fibonacci.out','a+')
def fib(n):
    a, b = 0, 1
    while b < n :
        print b,
        f.writelines( str(b))
        a, b = b, a+b
    f.writelines("\n")
    f.flush()
```

```
>>> reload(fibo)
<module 'fibo' from 'fibo.py'>
>>> from fibo import fib, f
>>> fib(30000)
1 1 2 3 5 8 13 21 34 55 89 ...
>>> fib(3000)
1 1 2 3 5 8 13 21 34 55 89 144 ...
>>> fib(30)
1 1 2 3 5 8 13 21
>>> fib(3)
1 1 2
>>>
```

```
[gidget:~/working/python/cs696] mthomas% cat fibonacci.out
112358132134558914423337761098715972584
1123581321345589144233
1123581321
112
```

Help(): Command-line PyDoc

```
>>> help()
```

```
help> Help on module os:
```

```
NAME
```

```
os - OS routines for Mac, DOS, NT, or Posix depending on what system we're on.
```

```
FILE
```

```
/System/Library/Frameworks/Python.framework/Versions/2.3/lib/python2.3/os.py
```

```
DESCRIPTION
```

```
This exports:
```

- all functions from posix, nt, os2, mac, or ce, e.g. unlink, stat, etc.
- os.path is one of the modules posixpath, ntpath, or macpath
- os.name is 'posix', 'nt', 'os2', 'mac', 'ce' or 'riscos'
- os.curdir is a string representing the current directory ('.' or ':')
- os.pardir is a string representing the parent directory ('..' or '::')
- os.sep is the (or a most common) pathname separator ('/' or ':' or '\\')
- os.extsep is the extension separator ('.' or '/')
- os.altsep is the alternate pathname separator (None or '/')
- os.pathsep is the component separator used in \$PATH etc
- os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
- os.defpath is the default search path for executables

OS File Methods

- Open
 - `F=open(path)`
- Read:
 - `S=F.read()`
 - `S=F.readline()`
 - `L=F.readlines()`
- Write:
 - `F.write(s)`
 - `F.writelines(sL)`
- Save
 - `F.flush()` <--outputs buffer contents
 - `F.close`
- Change file size
 - `F.truncate()`
 - `F.truncate(n)`
- Inside file:
 - `F.seek(...)`
 - `F.tell()` <--location
 - `F.fileno()` file descrip

Classes

- namespace: associated with objects
 - mapping from names to objects
 - Vary in time and persistence
- Attributes (any name following a `'.'`):
 - `range.reverse`, `reverse` is attribute
 - May be readable or writeable
 - Data and methods
- Scope: where a namespace accessible
 - Determined statically/context, accessed dynamically
 - Operations apply to local scope
- Class `MyClass`:
 - Works like the function `'def'` command
 - Can contain statements, vars, functions, etc.

Classes

```
import math
```

```
class Point:
```

```
    "PyDoc comment"
```

```
    print 'First class initialize statement'
```

```
    def __init__(self, x = 0, y = 0):
```

```
        print 'In constructor'
```

```
        self.x = x
```

```
        self.y = y
```

```
    print 'Second class initialize statement'
```

```
    def __add__(self, other):
```

```
        return Point(self.x + other.x, self.y + other.y)
```

```
    def distance(self):
```

```
        return math.sqrt(self.x * self.x + self.y * self.y)
```

```
    print '3rd class initialize statement'
```

```
>>> from myClass import Point
```

```
First class initialize statement
```

```
Second class initialize statement
```

```
3rd class initialize statement
```

```
>>> a=Point(1,2)
```

```
In constructor
```

```
>>> b=Point()
```

```
In constructor
```

```
>>> print a.distance()
```

```
2.2360679775
```

```
>>> print (a+a).distance()
```

```
In constructor
```

```
4.472135955
```

```
>>> print a
```

```
<myClass.Point instance at 0x63be8>
```

Some Terms

All attributes are public

All method attributes are virtual like Java

Python	C++	Java	Smalltalk
data attribute	data member	field	instance variable
method attribute	function member	method	instance method
class attribute	~static data member	~static field	class instance variable
classmethod			
staticmethod			

Different Data Attributes

Accessing

```
class DataDifferences:
    classAttribute = 10

    def __init__(self):
        self.data = 6
        whatIsThis = 'guess'

    def sampleAccess(self):
        classAttribute + self.data
```

```
print DataDifferences.classAttribute
```

```
example = DataDifferences()
print example.classAttribute
print example.data #error
```

```
print DataDifferences.data #error
```

Adding to an Object

```
def increase(x):  
    return x + 1  
  
class Empty:  
    pass
```

```
example = Empty()  
  
example.x = 5  
print example.x           #prints 5  
  
example.plus = increase  
print example.plus(5)     #prints 6  
  
different = Empty()  
print different.x         #runtime error
```

Deleting

```
class One:
    def __init__(self):
        self.x = 9

    def printer(self):
        print 'inside' , self.x
```

```
>>> ex=One()
>>> ex.x=5
>>> print ex.x
5
>>> ex.printer()
inside 5
>>> q=One()
>>> print q.x
9
>>> q.printer()
inside 9

>>> print One().x
9
>>> print ex.x
5
>>> del ex.x
>>> print One().x
9
>>> print ex.x
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: One instance has no attribute 'x'
>>>
```

Class attribute as default attribute value

```
class Ouch(object):  
    x = 10  
  
    def printer(self):  
        print self.x
```

You may want to avoid using same name for class and data attributes

```
>>> a=Ouch()  
>>> a.printer()  
10  
>>> a.x=5  
>>> a.printer()  
5  
>>> print Ouch.x  
10  
>>> b=Ouch()  
>>> b.printer()  
10  
>>>
```

Inheritance

```
class Parent:
    def __init__(self):
        self.x = 9
        self.z = 1

    def printer(self):
        print 'parent' , self.x, self.z

# BaseClassName is Parent
class Child(Parent):
    def __init__(self):
        Parent.__init__(self)
        self.y = 7
        self.z = 2

    def printer(self):
        Parent.printer(self)
        print 'child' , self.x, self.y, self.z
```

```
example = Child()
example.printer()
```

Output
parent 9 2
child 9 7 2

Multiple Inheritance (see O'rielly)

```
class Mother:
    def printer(self):
        print 'Mother'

class Father:
    def printer(self):
        print 'Father'

class Child2(Mother, Father):
    pass

class Child3(Father, Mother):
    pass
```

```
>>> c2=Child2()
>>> c2.printer()
Mother
>>> c3=Child3()
>>> c3.printer()
Father
>>>
```

New Classes - Python 2.2

<http://www.python.org/download/releases/2.2.3/descrintro/>

Subclassing built-in types
Static methods & class methods
Properties
super
__new__
Improved Metaclasses

```
New Classes
class C(object):
    pass

class D(C):
    pass
```

```
Classic Classes
class A:
    pass

class B(A):
    pass
```

super

```
class A(object):
    def __init__(self):
        print "A"
        super(A, self).__init__()

class B(A):
    def __init__(self):
        print "B"
        super(B, self).__init__()
```

B()

Output

B

A

Read about problems with Python's super at:
<http://fuhm.net/super-harmful/>

Static & Class Methods

```
class Example(object):
    staticVariable = 10

    def staticExample(x):
        print "static", x
        #Can not access staticVariable

    staticExample = staticmethod(staticExample)

    def classExample(cls,x):
        print 'class' , cls.__name__, x, cls.staticVariable

    classExample = classmethod(classExample)
```

```
a = Example()
Example.staticExample(1)
a.staticExample(2)
Example.classExample(3)
a.classExample(4)
```

Output

```
static 1
static 2
class Example 3 10
class Example 4 10
```

Private - sort of

```
class SortOfPrivate:
    __x = 0

    def __hiddenPrint(self):
        print 'Hidden'

    def printer(self):
        print self.__x,
        self.__hiddenPrint()
```

```
example = SortOfPrivate()
example.printer()
print example._SortOfPrivate__x
example._SortOfPrivate__hiddenPrint()
example.__hiddenPrint()
#error
```

Output

```
0 Hidden
0
Hidden
```

Standard Operators

```
class OverLoadExample:
    x = 0
    list = [1, 2, 3]
    def __del__(self):
        print 'Like a destructor'

    #convert to a string
    def __str__(self):
        return `self.x`

    #indexing operations
    def __getitem__(self, key):
        return self.list[key]

    def __setitem__(self, key, value):
        self.list[key] = value
```

```
a = OverLoadExample()
print a
a[1] = 5
print a[1]
del a
```

Output

```
0
5
Like a destructor
```

<http://docs.python.org/ref/customization.html>

Exceptions

```
try:  
    <block>  
except <name>:  
    <except block>  
except <name> , <data>:  
    <except block2>  
else:  
    <else block>
```

```
try:  
    <block>  
finally:  
    <finally block>
```

```
def myBad(list, index):  
    print list[index]  
  
try:  
    myBad([0, 1], 3)  
except IndexError:  
    print 'BadIndex'
```

<http://docs.python.org/lib/module-exceptions.html>

Defining an Exception

```
class SampleError(Exception):  
    pass
```

```
try:  
    raise SampleError  
except SampleError:  
    print 'I got it'  
else:  
    print 'No Error'
```

Modules

In file eiy.py

```
def whitney():  
    print 'whitney'
```

```
import eli  
eli.whitney()
```

```
from eli import whitney  
whitney()
```

```
from eli import *  
whitney()
```

How Python finds Modules

Python interpreter searches in order:

- Current directory

- Directories listed in environment variable PYTHONPATH

- Installation-dependent path for standard modules

__name__ and Modules

Module attribute

Set to '__main__' if file is run a program

Set to module name if file is imported as a module

File name Example.py

```
def hello():  
    print 'Hello'  
  
if __name__ == '__main__':  
    hello()
```

As Program

At 60->python nameExample.py
Hello

PyUnit for Unit Testing

file queue.py

```
class Queue:
    def __init__(self):
        self._elements = []

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

    def enqueue(self, a):
        self._elements.append(a)

    def dequeue(self):
        first = self._elements[0]
        self._elements = self._elements[1:]
        return first
```

```
import unittest
from queue import Queue

class TestQueue(unittest.TestCase):

    def testEnqueue(self):
        queue = Queue()
        self.assertEqual(queue.size(), 0)
        queue.enqueue(1)
        self.assertEqual(queue.size(), 1)

    def testDequeue(self):
        queue = Queue()
        queue.enqueue(1)
        queue.enqueue(2)
        self.assertEqual(queue.size(), 2)
        self.assertEqual(queue.dequeue(), 1)
        self.assertEqual(queue.size(), 1)
        self.assertEqual(queue.dequeue(), 2)
        self.assertEqual(queue.size(), 0)

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

Test Results

..

Ran 2 tests in 0.000s

OK

When a Test Fails

```
import unittest
from queue import Queue

class TestQueue(unittest.TestCase):

    def testEnqueue(self):
        queue = Queue()
        self.assertEqual(queue.size(), 42)

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

FAIL: testEnqueue (__main__.TestQueue)

Traceback (most recent call last):

File "/Users/whitney/Courses/683/Fall06/python/queueTest.py", line 8, in testEnqueue

self.assertEqual(queue.size(), 42)

AssertionError: 0 != 42

Ran 1 test in 0.002s

FAILED (failures=1)

setUp, tearDown

setUp() is run before each test method
tearDown() is run after each test method

```
import unittest
from queue import Queue

class TestQueue(unittest.TestCase):
    def setUp(self):
        self.queue = Queue()

    def testEnqueue(self):
        self.assertEqual(self.queue.size(), 0)
        self.queue.enqueue(1)
        self.assertEqual(self.queue.size(), 1)

    def testDequeue(self):
        self.queue.enqueue(1)
        self.queue.enqueue(2)
        self.assertEqual(self.queue.size(), 2)
        self.assertEqual(self.queue.dequeue(), 1)
        self.assertEqual(self.queue.size(), 1)

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

Useful Methods

`assert_(expression)`

<http://www.python.org/doc/current/lib/testcase-objects.html>

`failUnless(expression)`

`assertEqual(first, second[, msg])`

`failUnlessEqual(first, second[, msg])`

`assertNotEqual(first, second[, msg])`

`failIfEqual(first, second[, msg])`

`assertAlmostEqual(first, second[, places[, msg]])`

`failUnlessAlmostEqual(first, second[, places[, msg]])`

`assertNotAlmostEqual(first, second[, places[, msg]])`

`failIfAlmostEqual(first, second[, places[, msg]])`

`assertRaises(exception, callable, ...)`

`failUnlessRaises(exception, callable, ...)`

Testing Exceptions

```
import unittest
from queue import Queue
class TestQueue(unittest.TestCase):
    def setUp(self):
        self.queue = Queue()

    def testEnqueue(self):
        self.assertEqual(self.queue.size(), 0)
        self.queue.enqueue(1)
        self.assertEqual(self.queue.size(), 1)

    def testEmptyDequeue(self):
        self.assertRaises(IndexError, self.queue.dequeue)

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

Modules and Executable code

file eli.py

```
x = 0
y = [1, 2]
print 'Running module "eli"'

def whitney():
    print 'whitney'

def printValues():
    print x , y
```

```
import eli
eli.whitney()
from eli import y
print y
```

Output

```
Running module "eli"
whitney
[1, 2]
```

Reloading Modules

file eli.py

```
x = 0
y = [1, 2]
print 'Running module "eli"'

def whitney():
    print 'whitney'

def printValues():
    print x , y
```

```
import eli
eli.y[0] = 'cat'
eli.printValues()
```

```
reload(eli)
eli.printValues()
```

Output

```
Running module "eli"
0 ['cat', 2]
Running module "eli"
0 [1, 2]
```

Explain this

file eli.py

```
x = 0
y = [1, 2]
print 'Running module "eli"'

def whitney():
    print 'whitney'

def printValues():
    print x , y
```

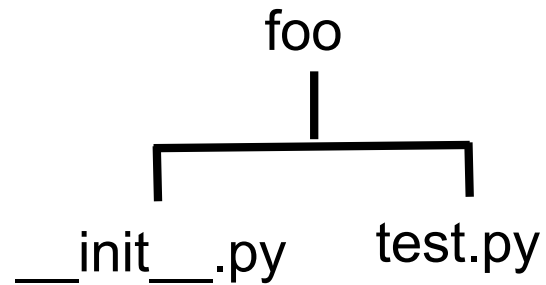
```
from eli import x, y, printValues
```

```
printValues()
y[0] = 'cat'
x = 'dog'
printValues()
```

Output

```
Running module "eli"
0 [1, 2]
0 ['cat', 2]
```

Packages - Nested Modules



In a directory in PYTHONPATH create a subdirectory, call it foo

In foo create an empty file:

`__init__.py`

Any .py file in foo is module

Comments on Assignment #1

<http://www-rohan.sdsu.edu/faculty/mthomas/courses/spring07/cs696/assignments/assignment1.html>