CS 683 Emerging Technologies
Fall Semester, 2006
Doc 3 Python - Classes, Modules, Unit tests
Sep 5, 2006

Reading Assignment

Aug 31. Chapters 1-4 of the Python Tutorial
Sept 5. Chapters 5-8 of the Python Tutorial
Sept 7. Chapters 9-11 of the Python Tutorial

Quiz & Assignment

First Quiz - Sept 12

Assignment 1 due - Sept 12

References

Python Tutorial, Guido van Rossum,
http://www.python.org/doc/current/tut/tut.html

Python Reference Manual, Guido van Rossum,
http://docs.python.org/ref/ref.html

Python Library Reference, Guido van Rossum,
http://docs.python.org/lib/lib.html

Learning Python, Lutz & Ascher, O'Reilly, 1999

Python Documentation

http://docs.python.org/

Python Documentation - Windows

Start > Programs >Python2.4 > Python Manuals

Same as http://docs.python.org/

Start > Programs >Python2.4 > Module Docs

(PyDoc)
Classes
Methods
Parameters

Command-line PyDoc

```
Al 20->python2.4
Python 2.4.3 (#1, Apr  7 2006, 10:54:33)
>>> help()
help> list
Help on class list in module __builtin__:

class list(object)
 |  list() -> new list
 |  list(sequence) -> new list initialized from sequence's items
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
 |  __contains__(...)
 |      x.__contains__(y) <==> y in x
 |
 |  __delitem__(...)
 |      x.__delitem__(y) <==> del x[y]
 |
```

Creating PyDocs

file: docExample.py

```
"this is a module level comment"

def adder(a,b):
        "Adder documentation"
        return a + b

def subtracter(a, b):
"""This is a multiline
comment
"""
return a - b
```

```
help> docExample
Help on module docExample:

NAME
    docExample - this is a module level comment

FILE
    /Users/whitney/Courses/683/Fall06/python/docExample.py

FUNCTIONS
    adder(a, b)
        Adder documentation

    subtracter(a, b)
        This is a multiline
        comment
```

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'
```

```
print 'start'
a = Point(1,2)
b = Point()
print a.distance()
print (a + a).distance()
print a
```

Output

```
First class initialize statement
Second class initialize statement
3rd class initialize statement
start
In constructor
In constructor
2.2360679775
In constructor
4.472135955
<__main__.Point instance at 0x73238>
```

Some Terms

All attributes are public

All method attributes are virtual like Java

Different Data Attributes

```
class DataDifferences:
    classAttribute = 10

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

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

Accessing

```
print DataDifferences.classAttribute

example = DataDifferences()
print example.classAttribute
print example.data

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
```

```
example = One()

example.x = 5
print example.x          # 5
example.printer()        # Inside 5

del example.x
print One().x            # 9
print example.x          # error
```

Class attribute as default attribute value

```
class Ouch(object):
    x = 10

    def printer(self):
        print self.x
```

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

Output

```
10
5
10
10
```

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

Inheritance

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

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

```
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

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

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

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

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

Output

```
Mother
```

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 ely.py

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

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

```
from ely import whitney
whitney()
```

```
from ely 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 nameExample.py

```
def hello():
    print 'Hello'

if __name__ == '__main__':
    hello()
```

As Program

```
Al 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

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

assert_(expression)
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()
```