CS 683 Emerging Technologies
Fall Semester, 2005
Doc 3 Python Classes
Contents

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

Classes

```
import math

class Point:
    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)

    print '3rd class initialize statement'

    def distance(self):
        return math.sqrt(self.x * self.x + self.y * self.y)

    print '4th class initialize statement'
```

All class members are public

All member functions are virtual like Java

Sample Use

```
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
4th class initialize statement
In constructor
In constructor
2.2360679775
In constructor
4.472135955
<__main__.Point instance at 0x73238>
```

Initializing a Data member

```
class ClassData:
    x = 5
    print 'x= ' , x
    def printer(self):
        print self.x
```

```
example = ClassData()
example.printer()
example.x = 3
example.printer()
second = ClassData()
second.printer()
```

Output

```
x=  5
5
3
5
3
```

Adding attributes 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 Attributes

```
class One:
    x = 9

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

example = One()
example.x = 5
print example.x
example.printer()
del example.x
print example.x
```

Output

```
5
inside 5
9
```

Inheritance

```
class Parent:
    x = 9
    z =1
    def printer(self):
        print 'parent' , self.x, self.z

class Child(Parent):
    y = 7
    z = 2
    def printer(self):
        Parent.printer(self)
        print 'child' , self.x, self.y, self.z
```

Sample Usage

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

Output

```
parent 9 2
child 9 7 2
```

Multiple Inheritance

```
class Mother:
    x = 0
    z = 1
    def printer(self):
        print 'Mother' , self.x, self.z

class Father:
    y = 7
    z = 2
    def printer(self):
        Parent.printer(self)
        print 'Father' , self.y, self.z

class Child(Mother, Father):
    pass

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

Output

```
Mother 0 1
```

Sort of Private Identifiers

```
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          #valid
example._SortOfPrivate__hiddenPrint()    #valid
example.__hiddenPrint()                  #error
```

Identifiers starting with two underscores are name mangled to simulate private methods and fields

Overloading Standard Operators

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

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

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

    def __setitem__(self, key, value): #indexing operation
        self.list[key] = value

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

Output

```
0
5
Like a destructor
```

Overloading Details

Iterators

Used by the for statement

Iterator methods

- next()
- __iter__()

Example

```
>>> a = 'cat'
>>> iterator = iter(a)
>>> iterator
<iterator object at 0x6ff10>
>>> iterator.next()
'c'
>>> iterator.next()
'a'
>>> iterator.next()
't'
>>> iterator.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
StopIteration
```

Iterator Class
Example (from Python Tutorial)

```
class Reverse:
    "Iterator for looping over a sequence backwards"
    def __init__(self, data):
        self.data = data
        self.index = len(data)
    def __iter__(self):
        return self
    def next(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]
```

Sample Usage

```
for c in Reverse('cat'):
    print c
```

Output

```
t
a
c
```

(Example from Python tutorial)

Generators

Used to create iterators

Resumable functions

Example (from Python tutorial)

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]

for c in reverse('cat'):
    print c
```

Object Iteration Example

```
class MixedType:
    x = 0
    list = [1, 2, 3]

    def __iter__(self):
        return self.__mixedTypeGenerator()

    def __mixedTypeGenerator(type):
        yield type.x
        for a in type.list:
            yield a

a = MixedType()
for x in a:
    print x
```

Output

```
0
1
2
3
```

Object Dictionaries

```
class ObjectDictionary:

    def createX(self):
        self.x = 5

    def createY(self):
        self.y = 10

a = ObjectDictionary()
print a.__dict__
a.createX()
print a.__dict__
a.createY()
print a.__dict__
a.__dict__['sam'] = 20
print a.__dict__
a.pete = 30
print a.__dict__
```

Output

```
{}
{'x': 5}
{'y': 10, 'x': 5}
{'y': 10, 'x': 5, 'sam': 20}
{'y': 10, 'x': 5, 'pete': 30, 'sam': 20}
```

Doc Strings

Example (from Python manual)

```
def function():
    "function doc string"
    return 5

class DocStringExample:
    "This is the class doc string"

    def method(self):
        "method doc string"
        print 'hi'

print function.__doc__
print DocStringExample.__doc__
print DocStringExample.method.__doc__
```

Output

```
function doc string
This is the class doc string
method doc string
```

doctest

```
"""
This is the "example" module.
>>> factorial(5)
120
"""

def factorial(n):
    """Return the factorial of n, an exact integer >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000L
    """
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result

def _test():
    import doctest
    doctest.testmod()

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

Running the Tests in the Comments

Place the code in a file - examples.py

Run the module

```
Al 21->python examples.py
```

If all test pass, no output

If a test fails list of failures

Example of No failures

```
Al 21->python examples.py
Al 22->
```

Example of a Failure

```
Al 24->python examples.py
**********************************************************************
Failure in example: factorial(5)
from line #2 of __main__
Expected: 121
Got: 120
**********************************************************************
1 items had failures:
   1 of   1 in __main__
***Test Failed*** 1 failures.
Al 25->
```

More details on doc test