Introduction to Python Lecture-2

Gaurav Saxena
Scientific Officer
VECC, Kolkata

Contents

An Example

```
# reads in the text file whose name is specified on the command line,
# and reports the number of lines and words

import sys

def checkline():
    global l
    global wordcount
    w = l.split()
    wordcount += len(w)

wordcount = 0
f = open(sys.argv[1])
flines = f.readlines()
linecount = len(flines)
for l in flines:
    checkline()
print linecount, wordcount
```

An Example

- File is **x** with the contents

```
This is an
Example of an
text file.
```

- Output is

```
$python example.py x
5 8
```

Command-Line Arguments

- "**import sys**" includes a module (i.e. library) named **sys**.
- Need to explicitly load **sys**.
- List **argv** is member variable of **sys**.
- Analogous to argv in C/C++
- **sys.argv[1]** will be the string 'x'
- To convert the argument into **int** or **float** we use **int()** and **float()** respectively.

Introduction to File Manipulation

- The function **open()** is similar to the one in C/C++.
- `f = open(sys.argv[1])`, created an object of file class, and assigned it to f
- The **readlines()** function of the file class returns a list consisting of the lines in the file.
- Each line is a string, and that string is one element of the list.
- In this case

```
['','This is an','example of a','text file','']
```

Lack of Declaration

- Variables are not declared in Python.
- A variable is created when the first assignment to it is executed.
- the variable flines does not exist until the statement `flines = f.readlines()` is executes

Locals Vs. Globals

- Python does not really have global variables in the sense of C/C++.
- But for now, for a single source file, its pretty much similar to that of C/C++.
- Python tries to infer the scope of a variable from its position in the code.
- If a function includes any code which assigns to a variable, then that variable is assumed to be local.
- In the example, Python would assume that **l** and **wordcount** are local to **checkline()** if we don’t inform it otherwise.

Built-In Functions

- len()
  - Returns the number of elements in a list.
- readlines()
  - Returns a list in which each element consisted of one line of the file
- split()
  - splits a string into a list of words

Keyboard Input

- name = raw_input('enter a name: ')   #input() in case of python 3

```
>>> name = input('Enter your name:')
Enter your name:Gaurav Saxena
>>> name
'Gaurav Saxena\r'
```

- Alternatively

```
>>> import sys
>>> z = sys.stdin.readlines()
abc
de
f
>>> z
['abc\n', 'de\n', 'f\n']
```

Object-Oriented Programming: Overview

- **Class:** A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable:** A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables aren't used as frequently as instance variables are.
- **Data member:** A class variable or instance variable that holds data associated with a class and its objects.
- **Function overloading:** The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects (arguments) involved.
- **Instance variable:** A variable that is defined inside a method and belongs only to the current instance of a class.

Object-Oriented Programming: Overview

- **Inheritance :** The transfer of the characteristics of a class to other classes that are derived from it.
- **Instance:** An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- **Instantiation :** The creation of an instance of a class.
- **Method :** A special kind of function that is defined in a class definition.
- **Object :** A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- **Operator overloading:** The assignment of more than one function to a particular operator.

Creating Classes

```
class Employee:
  'Common base class for all employees‘
  empCount = 0
  def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
  def displayCount(self):
      print "Total Employee %d" %
  Employee.empCount
  def displayEmployee(self):
      print "Name : ", self.name, ", Salary: ",
  self.salary
```

Creating instance objects:

```
emp1 = Employee("Zara", 2000)
emp2 = Employee("Manni", 5000)
```

Accessing attributes:

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" %
  Employee.empCount
```

- add, remove, or modify attributes of classes and objects at any time:

```
emp1.age = 7 # Add an 'age' attribute.
emp1.age = 8 # Modify 'age' attribute.
del emp1.age # Delete 'age' attribute.
```

Accessing attributes:

```
hasattr(emp1, 'age') # Returns true if 'age'
  attribute exists

getattr(emp1, 'age') # Returns value of
  'age' attribute

setattr(emp1, 'age', 8) # Set attribute
  'age' at 8

delattr(empl, 'age') # Delete attribute
  'age'
```

Built-In Class Attributes:

- __dict__ : Dictionary containing the class's namespace.
- __doc__ : Class documentation string, or None if undefined.
- __name__: Class name.
- __module__: Module name in which the class is defined. This attribute is "__main__" in interactive mode.
- __bases__ : A possibly empty tuple containing the base classes, in the order of their occurrence in the base class list.

Built-In Class Attributes:

```
Employee.__doc__: Common base class for all
  employees
Employee.__name__: Employee
Employee.__module__: __main__
Employee.__bases__: ()
Employee.__dict__: {'__module__':
  '__main__', 'displayCount': <function
  isplayCount at 0xb7c84994>, 'empCount': 2,
  'displayEmployee': <function
  displayEmployee at 0xb7c8441c>, '__doc__':
  'Common base class for all employees',
  '__init__': <function __init__ at
  0xb7c846bc>}
```

Destructors

```
class Point:
  def __init( self, x=0, y=0):
       self.x = x
       self.y = y
  def __del__(self):
       class_name = self.__class__.__name__
       print class_name, "destroyed"
pt1 = Point()
pt2 = pt1
pt3 = pt1
print id(pt1), id(pt2), id(pt3) # prints the ids of the
  objects
del pt1
del pt2
del pt3
```

Destructors

```
3083401324 3083401324 3083401324
Point destroyed
```

- When an object's reference count reaches zero, Python collects it automatically

Class Inheritance:

```
class SubClassName (ParentClass1[,
  ParentClass2, ...]):
  'Optional class documentation string'
  class_suite
```

Class Inheritance:

```
class Parent: # define parent class
  parentAttr = 100
  def __init__(self):
      print "Calling parent constructor"
  def parentMethod(self):
      print 'Calling parent method'
  def setAttr(self, attr):
      Parent.parentAttr = attr
  def getAttr(self):
      print "Parent attribute :", Parent.parentAttr
```

Class Inheritance:

```
class Child(Parent): # define child
  class
  def __init__(self):
      print "Calling child constructor"
  def childMethod(self):
      print 'Calling child method'
```

Class Inheritance:

```
c = Child() # instance of child
c.childMethod() # child calls its method
c.parentMethod() # calls parent's method
c.setAttr(200) # again call parent's method
c.getAttr() # again call parent's method
```

- Output

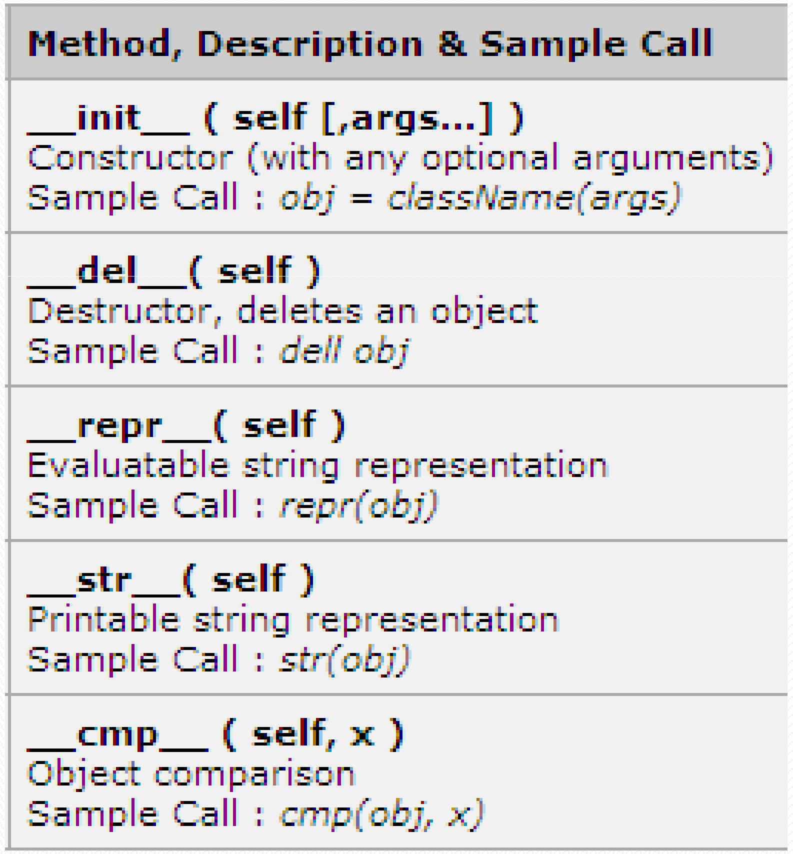
```
Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200
```

```
#!/usr/bin/python

class Parent:          # define parent class
   parentAttr = 100
   def __init__(self):
      print "Calling parent constructor"

   def parentMethod(self):
      print 'Calling parent method'

   def setAttr(self, attr):
      Parent.parentAttr = attr

   def getAttr(self):
      print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
   def __init__(self):
      print "Calling child constructor"

   def childMethod(self):
      print 'Calling child method'

c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.getAttr()          # again call parent's method
```

This would produce following result:

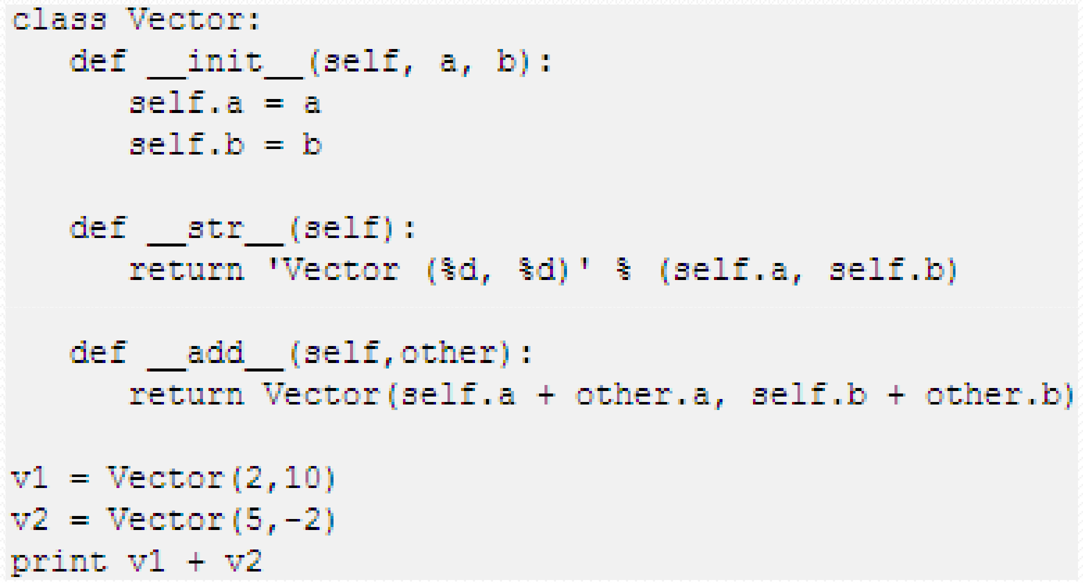
```
Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200
```

Base Overloading Methods:



Method, Description & Sample Call
__init__ (self [,args...]) Constructor (with any optional arguments) Sample Call : *obj = className(args)*
__del__(self) Destructor, deletes an object Sample Call : *dell obj*
__repr__(self) Evaluatable string representation Sample Call : *repr(obj)*
__str__(self) Printable string representation Sample Call : *str(obj)*
__cmp__ (self, x) Object comparison Sample Call : *cmp(obj, x)*

Overloading Operators:



```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print v1 + v2
```

- Output

```
Vector(7,8)
```

Data Hiding:

```
class JustCounter:
   __secretCount = 0

   def count(self):
      self.__secretCount += 1
      print self.__secretCount

counter = JustCounter()
counter.count()
counter.count()
print counter.__secretCount
```

- Output

```
1
2
Traceback (most recent call last):
  File "test.py", line 12, in <module>
    print counter.__secretCount
AttributeError: JustCounter instance has no attribute '__secretCount'
```

Data Hiding:

- Python protects those members by internally changing the name to include the class name
- We can access such attributes as *object._className__attrName*
- Ex.

```
print counter._JustCounter__secretCount


2
```

```
class textfile:
   ntfiles = 0  # count of number of textfile objects
   def __init__(self,fname):
      textfile.ntfiles += 1
      self.name = fname  # name
      self.fh = open(fname)   # handle for the file
      self.lines = self.fh.readlines()
      self.nlines = len(self.lines)  # number of lines
      self.nwords = 0  # number of words
      self.wordcount()
   def wordcount(self):
      "finds the number of words in the file"
      for l in self.lines:
         w = l.split()
         self.nwords += len(w)
   def grep(self,target):
      "prints out all lines containing target"
      for l in self.lines:
         if l.find(target) >= 0:
            print l

a = textfile('x')
b = textfile('y')
print "the number of text files open is", textfile.ntfiles
print "here is some information about them (name, lines, words):"
for f in [a,b]:
   print f.name,f.nlines,f.nwords
a.grep('example')
```

Example

- Output

```
python tfe.py
the number of text files opened is 2
here is some information about them
  (name, lines, words):
x 5 8
y 2 5
example of a
```

Files I/O

- Reading Keyboard Input:
  - The *input* Function:

```
str = input("Enter your input: ");
print "Received input is : ", str
```

  - Output

```
Enter your input: [x*5 for x in
  range(2,10,2)]
Recieved input is : [10, 20, 30, 40]
```

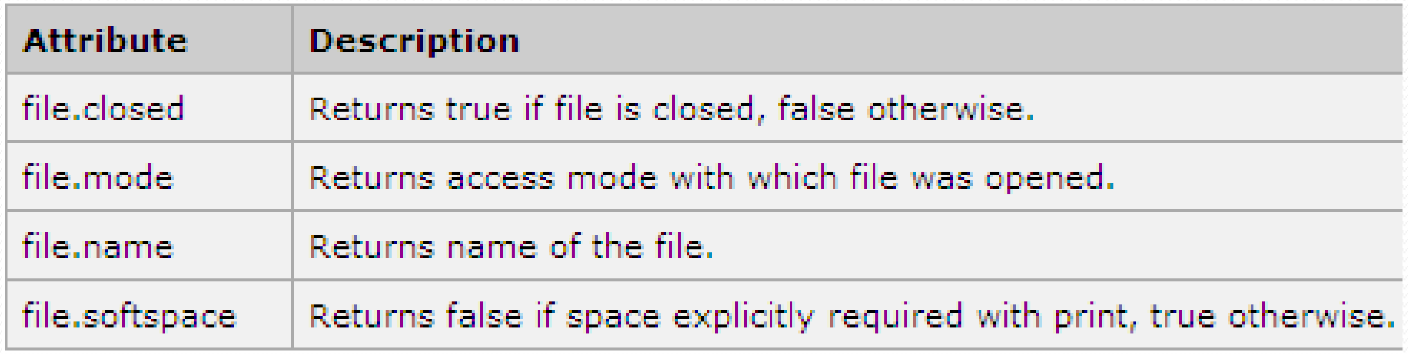
Files I/O

- Opening and Closing Files:
  - The *open* Function:

  file object = open(file_name [, access_mode][, buffering])
  - **file_name:** The file_name argument is a string value that contains the name of the file that you want to access.
  - **access_mode:** The access_mode determines the mode in which the file has to be opened ie. read, write append etc
  - **buffering:** For buffering.
  - r, rb, r+, rb+, w, wb, w+, wb+, a, ab, a+, ab+

Files I/O

- The *file* object atrributes:



Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

- The *close()* Method:

fileObject.close();

Files I/O

- Reading and Writing Files:
  - The *write()* Method:

```
# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great
  language.\nYeah its great!!\n");
# Close opend file
fo.close()
```

  - The *read()* Method:

```
str = fo.read(10);
```

Files I/O

- Renaming and Deleting Files:
  - The rename() Method:

```
import os # Rename a file from test1.txt
  to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

  - The *delete()* Method:

```
os.delete(file_name)
```