Python Programming

IOE 373 Lecture 15

Topics

I. Files
II. Lists
III. Dictionaries

I. Opening a File

- Before we can read the contents of the file, we must tell Python which file we are going to work with and what we will be doing with the file
- This is done with the open() function
- open() returns a “file handle” - a variable used to perform operations on the file
- Similar to “File -> Open” in a Word Processor

Using open()

- handle = open(filename, mode)
  - returns a handle use to manipulate the file
  - filename is a string
  - mode is optional and should be 'r' if we are planning to read the file and 'w' if we are going to write to the file

```
fhand = open('mbox.txt', 'r')
```

The newline Character

- We use a special character called the "newline" to indicate when a line ends
- We represent it as \n in strings
- Newline is still one character - not two

```
>>> stuff = 'Hello\nWorld!'
>>> stuff
'Hello\nWorld!'
>>> print (stuff)
Hello
World!
>>> stuff = 'X\nY'
>>> print (stuff)
X
Y
>>> len(stuff)
3
```

File Processing

- A text file has newlines at the end of each line

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008\n
Return-Path: <postmaster@collab.sakaiproject.org>\n
Date: Sat, 5 Jan 2008 09:12:18 -0500\n
To: source@collab.sakaiproject.org\n
From: stephen.marquard@uct.ac.za\n
Subject: [sakai] svn commit: r39772 - content/branches/\n
\n
Details: http://source.sakaiproject.org/viewsvn/?view=rev&rev=39772\n
```

File Handle as a Sequence

- A file handle open for read can be treated as a sequence of strings where each line in the file is a string in the sequence
- We can use the for statement to iterate through a sequence
- Remember - a sequence is an ordered set

```
xfile = open('mbox.txt')
for cheese in xfile:
    print (cheese)
```

Counting Lines in a File

- Open a file read-only
- Use a for loop to read each line
- Count the lines and print out the number of lines

```
fhand = open('mbox.txt')
count = 0
for line in fhand:
    count = count + 1
print ('Line Count:', count)
```

```
$ python open.py
Line Count: 132045
```

Reading the *Whole* File

- We can read the whole file (newlines and all) into a single string

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print (len(inp))
94626
>>> print (inp[:20])
From stephen.marquar
```

Searching Through a File

- We can put an **if** statement in our for loop to only print lines that meet some criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    if line.startswith('From:') :
        print (line)
```

OOPS!

What are all these blank lines doing here?

```
From: stephen.marquard@uct.ac.za

From: louis@media.berkeley.edu

From: zqian@umich.edu

From: rjlowe@iupui.edu
...
```

OOPS!

- Each line from the file has a newline at the end
- The print statement adds a newline to each line

```
From: stephen.marquard@uct.ac.za\n
\n
From: louis@media.berkeley.edu\n
\n
From: zqian@umich.edu\n
\n
From: rjlowe@iupui.edu\n
\n
...
```

Searching Through a File (fixed)

- We can strip the whitespace from the right-hand side of the string using **rstrip()** from the string library
- The newline is considered "white space" and is stripped

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.startswith('From:') :
        print (line)
```

From: stephen.marquard@uct.ac.za
From: louis@media.berkeley.edu
From: zqian@umich.edu
From: rjlowe@iupui.edu
....

Skipping with continue

- We can conveniently skip a line by using the continue statement

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:') :
        continue
    print (line)
```

Using in to select lines

- We can look for a string anywhere in a line as our selection criteria

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not '@uct.ac.za' in line :
        continue
    print (line)
```

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
X-Authentication-Warning: set sender to stephen.marquard@uct.ac.za using -f
From: stephen.marquard@uct.ac.za
Author: stephen.marquard@uct.ac.za
From david.horwitz@uct.ac.za Fri Jan  4 07:02:32 2008
X-Authentication-Warning: set sender to david.horwitz@uct.ac.za using -f...
```

Prompt for File Name

```
fname = input('Enter the file name:  ')
fhand = open(fname)
count = 0
for line in fhand:
    if line.startswith('Subject:') :
        count = count + 1
print ('There were', count, 'subject lines
in', fname)
```

Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

Enter the file name: mbox-short.txt
There were 27 subject lines in mbox-short.txt

Bad File Names

```
fname = input('Enter the file name:  ')
try:
    fhand = open(fname)
    flag=0
except:
    print ('File cannot be opened:', fname)
    flag=1

if flag==0:
    count = 0
    for line in fhand:
        if line.startswith('Subject:') :
            count = count + 1
    print ('There were', count, 'subject lines in', fname)
```

Enter the file name: mbox.txt
There were 1797 subject lines in mbox.txt

Enter the file name: na na boo boo
File cannot be opened: na na boo boo

II. Lists - List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object - even another list
- A list can be empty

```
>>> print ([1, 24, 76])
[1, 24, 76]
>>> print (['red', 'yellow',
'blue'])
['red', 'yellow', 'blue']
>>> print (['red', 24, 98.6])
['red', 24, 98.599999999999994]
>>> print ([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print ([])
[]
```

Lists and definite loops

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print ('Happy New Year:', friend)
print('Done!')
```

Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!

Looking Inside Lists

- Just like strings, we can get at any single element in a list using an index specified in square brackets

Joseph	Glenn	Sally
0	1	2

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print (friends[1])
Glenn
>>>
```

Lists are Mutable

- Strings are “immutable” - we cannot change the contents(e.g. individual characters) of a string - we must make a new string to make any change
- Lists are “mutable’ - we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> x = fruit.lower()
>>> print (x)
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print (lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print (lotto)
[2, 14, 28, 41, 63]
```

How Long is a List?

- The len() function takes a list as a parameter and returns the number of elements in the list
- Actually len() tells us the number of elements of any set or sequence (such as a string...)

```
>>> greet = 'Hello Bob'
>>> print (len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print (len(x))
4
>>>
```

Using the range function

- The range function returns a list of numbers that range from zero to one less than the parameter
- We can construct an index loop using for and an integer iterator

```
>>> print (range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print (len(friends))
3
>>> print (range(len(friends)))
[0, 1, 2]
>>>
```

Equivalent Loops...

```
friends = ['Joseph', 'Glenn', 'Sally']

for friend in friends :
    print ('Happy New Year:',  friend)

for i in range(len(friends)) :
    friend = friends[i]
    print ('Happy New Year:',  friend)
```

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print (len(friends))
3
>>> print (range(len(friends)))
[0, 1, 2]
>>>
```

Happy New Year: Joseph

Happy New Year: Glenn

Happy New Year: Sally

Concatenating lists using +

- We can create a new list by adding two existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print (c)
[1, 2, 3, 4, 5, 6]
>>> print (a)
[1, 2, 3]
```

Lists can be sliced using :

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Remember: *Just like in strings*, the second number is "up to but not including"

List Methods

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['append', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

http://docs.python.org/tutorial/datastructures.html

Building a List from Scratch

- We can create an empty list and then add elements using the append method
- The list stays in order and new elements are added at the end of the list

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print (stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print (stuff)
['book', 99, 'cookie']
```

Is Something in a List?

- Python provides two operators that let you check if an item is in a list
- These are logical operators that return **True** or **False**
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

A List is an Ordered Sequence

- A list can hold many items and keeps those items in the order until we do something to change the order
- A list can be sorted (i.e., change its order)
- The sort method means “sort yourself”

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort()
>>> print (friends)
['Glenn', 'Joseph', 'Sally']
>>> print (friends[1])
Joseph
>>>
```

Built-in Functions and Lists

- There are a number of functions built into Python that take lists as parameters
- Remember the loops we built? These are much simpler.

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print (len(nums))
6
>>> print (max(nums))
74
>>> print (min(nums))
3
>>> print (sum(nums))
154
>>> print (sum(nums)/len(nums))
25
```

It's easier with a list…

```
numlist = list()
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    numlist.append(value)

average = sum(numlist) / len(numlist)
print ('Average:', average)
```

Enter a number: 3

Enter a number: 9

Enter a number: 5

Enter a number: done

Average: 5.66666666667

VS.

```
total = 0
count = 0
while True :
    inp = input('Enter a number: ')
    if inp == 'done' : break
    value = float(inp)
    total = total + value
    count = count + 1

average = total / count
print ('Average:', average)
```

Strings and Lists

```
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print (stuff)
['With', 'three', 'words']
>>> print (len(stuff))
3
>>> print (stuff[0])
With
```

```
>>> print (stuff)
['With', 'three', 'words']
>>> for w in stuff :
...     print (w)
...
With
Three
Words
>>>
```

Split breaks a string into parts and produces a list of strings. We think of these as words. We can access a particular word or loop through all the words.

Delimiters

When you do not specify a delimiter, multiple spaces are treated like *one* delimiter -

You can specify what delimiter character to use in the splitting

```
>>> line = 'A lot                    of spaces'
>>> etc = line.split()
>>> print (etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first;second;third'
>>> thing = line.split()
>>> print (thing)
['first;second;third']
>>> print (len(thing))
1
>>> thing = line.split(';')
>>> print (thing)
['first', 'second', 'third']
>>> print (len(thing))
3
>>>
```

Example

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print (words[2])
```

Sat

Fri

Fri

Fri

...

```
>>> line = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> words = line.split()
>>> print (words)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>
```

The Double Split Pattern

- Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
pieces = email.split('@')
print (pieces[1])
```

```
stephen.marquard@uct.ac.za
['stephen.marquard', 'uct.ac.za']
'uct.ac.za'
```

III. Dictionaries: Two Types of Collections

- List
  - A linear collection of values that stay in order
- Dictionary
  - A “bag” of values, each with its own label

Dictionaries

- Dictionaries are Python's most powerful data collection
- Dictionaries allow us to do fast database-like operations in Python
- Dictionaries have different names in different languages
  - Associative Arrays - Perl / PHP
  - Properties or Map or HashMap - Java
  - Property Bag - C# / .Net

http://en.wikipedia.org/wiki/Associative_array

Dictionaries

- Lists index their entries based on the position in the list
- Dictionaries are like bags - no order
- So we index the things we put in the dictionary with a "lookup tag"

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print (purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print (purse['candy'])
3
>>> purse['candy'] = purse['candy'] + 2
>>> print (purse)
{'money': 12, 'tissues': 75, 'candy': 5}
```

Comparing Lists and Dictionaries

- Dictionaries are like lists except that they use keys instead of numbers to look up values

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print (lst)
[21, 183]
>>> lst[0] = 23
>>> print (lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print (ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print (ddd)
{'course': 182, 'age': 23}
```

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print (lst)
[21, 183]
>>> lst[0] = 23
>>> print (lst)
[23, 183]
```

List

Key	Value
[0]	21
[1]	183

lst

```
>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print (ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print (ddd)
{'course': 182, 'age': 23}
```

Dictionary

Key	Value
['course']	182
['age']	21

ddd

Dictionary Literals (Constants)

- Dictionary literals use curly braces and have a list of keys : value pairs
- You can make an empty dictionary using empty curly braces

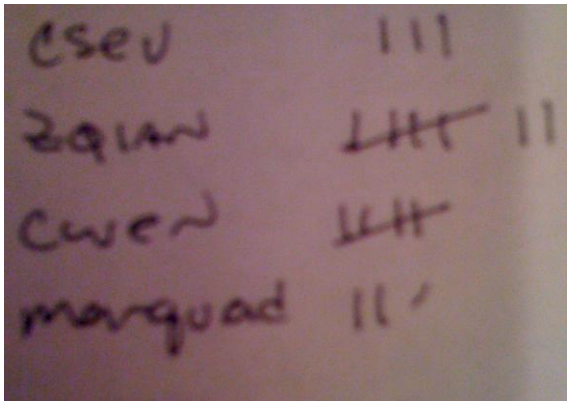
```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print (jjj)
{'jan': 100, 'chuck': 1, 'fred': 42}
>>> ooo = { }
>>> print (ooo)
{}
>>>
```

Many Counters with a Dictionary

- One common use of dictionary is counting how often we "see" something

```
>>> ccc = dict()
>>> ccc['csev'] = 1
>>> ccc['cwen'] = 1
>>> print (ccc)
{'csev': 1, 'cwen': 1}
>>> ccc['cwen'] = ccc['cwen'] + 1
>>> print (ccc)
{'csev': 1, 'cwen': 2}
```

Key Value



Dictionary Tracebacks

- It is an **error** to reference a key which is not in the dictionary
- We can use the **in** operator to see if a key is in the dictionary

```
>>> ccc = dict()
>>> print (ccc['csev'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'csev'

>>> print ('csev' in ccc)
False
```

When we see a new name

- When we encounter a new name, we need to add a new entry in the dictionary and if this is the second or later time we have seen the name, we simply add one to the count in the dictionary under that name

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    if name not in counts:
       counts[name] = 1
    else :
        counts[name] = counts[name] + 1
print (counts)
```

{'csev': 2, 'zqian': 1, 'cwen': 2}

The **get** method for dictionaries

- This pattern of checking to see if a key is already in a dictionary and assuming a default value if the key is not there is so common, that there is a method called get() that does this for us

```
if name in counts:
    x = counts[name]
else :
    x = 0
```

```
x = counts.get(name, 0)
```

Default value if key does not exist

{'csev': 2, 'zqian': 1, 'cwen': 2}

Simplified counting with get()

- We can use get() and provide a default value of zero when the key is not yet in the dictionary - and then just add one

```
counts = dict()
names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
for name in names :
    counts[name] = counts.get(name, 0) + 1
print (counts)
```

Default

{'csev': 2, 'zqian': 1, 'cwen': 2}

Counting Pattern

```
counts = dict()
print ('Enter a line of text:')
line = input('')

words = line.split()

print ('Words:', words)

print ('Counting... ')
for word in words:
    counts[word] = counts.get(word,0) + 1
print ('Counts', counts)
```

The general pattern to count the words in a line of text is to split the line into words, then loop through the words and use a dictionary to track the count of each word independently.

Counting Words

```
python wordcount.py
Enter a line of text:
the clown ran after the car and the car ran into the tent
and the tent fell down on the clown and the car

Words: ['the', 'clown', 'ran', 'after', 'the', 'car',
'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and',
'the', 'tent', 'fell', 'down', 'on', 'the', 'clown',
'and', 'the', 'car']
Counting…

Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1,
'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7,
'tent': 2}
```

Definite Loops and Dictionaries

- Even though dictionaries are not stored in order, we can write a **for** loop that goes through all the entries in a dictionary - actually it goes through all of the keys in the dictionary and looks up the values

```
>>> counts = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> for key in counts:
...     print (key, counts[key])
...
jan 100
chuck 1
fred 42
>>>
```

Retrieving lists of Keys and Values

- You can get a list of keys, values, or items (both) from a dictionary

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42, 'jan': 100}
>>> print (list(jjj))
['jan', 'chuck', 'fred']
>>> print (jjj.keys())
['jan', 'chuck', 'fred']
>>> print (jjj.values())
[100, 1, 42]
>>> print (jjj.items())
[('jan', 100), ('chuck', 1), ('fred', 42)]
>>>
```

Two Iteration Variables

- We loop through the key-value pairs in a dictionary using *two* iteration variables
- Each iteration, the first variable is the key and the second variable is the corresponding value for the key

```
>>> jjj = { 'chuck' : 1 , 'fred' : 42,
'jan': 100}
>>> for aaa,bbb in jjj.items() :
...          print (aaa, bbb)
...
jan 100
chuck 1
fred 42
>>>
```

aaa	bbb
[jan]	100
[chuck]	1
[fred]	42

```
name = input('Enter file:')
handle = open(name)
text = handle.read()
words = text.split()

counts = dict()
for word in words:
   counts[word] = counts.get(word,0) + 1

bigcount = None
bigword = None
for word,count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

print (bigword, bigcount)
```

```
python words.py
Enter file: words.txt
to 16
```

```
python words.py
Enter file: clown.txt
the 7
```