PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Lab 10 File IO

Topics

- File I/O
- Module: `os`
- Walking A Directory

v.4.0

lab09_1.py

```
#!/usr/bin/env python
"""lab09.py Dictionary implementation for demonstrating a dictionary

   Edit the program so that is has another choice in the menu:
   (d)efinitions.

   This new option will print out the dictionary alphabetically by
   the meanings. """

from py_dict import *  # Careful of this!

def ListDefinitions():
    """Prints out the dictionary, alphabetically by the meanings"""
    defs = []     # or:  defs = [(v, k) for (k, v) in py_dict.items()]
    for k, v in py_dict.items():
        defs += [(v, k)]
    defs.sort()
    for (v, k) in defs:
        print v, ':', k


def ListDefinitions():
    """Prints out the dictionary, alphabetically by the meanings --
implemented via a sort function."""
    def ValueKey(x):
        return py_dict[x]

    for each in sorted(py_dict, key=ValueKey):
        print py_dict[each], ':', each

def main():
    """Runs the user interface for dictionary manipulation."""
    choices = {'add': CollectEntries, 'definitions': ListDefinitions,
               'find': FindDefinitions, 'print': PrintEntries}
    prompt = MakePrompt(choices)
    while True:
        raw_choice = raw_input(prompt)
        if not raw_choice:
            break
        given_choice = raw_choice[0].lower()
        for maybe_choice in choices:
            if maybe_choice[0] == given_choice:
                choices[maybe_choice]()
                break
```

```
45         else:
46             print '%s is not an acceptible choice.' % raw_choice
47
48 if __name__ == '__main__':
49     main()
50 """
51 $ lab09.py
52 Choose (a)dd, (d)efinitions, (f)ind, (p)rint (enter to quit) d
53 an object used to access a value in a dictionary : key
54 break out of a loop and skip the else : break
55 do nothing : pass
56 go to the next iteration of the loop : continue
57 Choose (a)dd, (d)efinitions, (f)ind, (p)rint (enter to quit)
58 $"""
```

file1.py

```
 1 #!/usr/bin/env python
 2 """Demonstrates reading a file line by line."""
 3
 4 def PrintFile(f_name):
 5     open_file = open(f_name)
 6     for line in open_file:
 7         print line,
 8     open_file.close()
 9
10 def main(f_name):
11     PrintFile(f_name)
12
13 if __name__ == '__main__':
14     main("ram.tzu")
15
16 """
17 $ file1.py
18 Ram Tzu know this:
19 When God wants you to do something,
20 you think it's your idea.
21 $
22 """
```

file2.py

```
 1 #!/usr/bin/env python
 2 """Demonstrates the 'finally' clause.  It happens whether or
 3 not there's an exception and it passes the exception up to the
 4 surrounding try/except.
 5
 6 *  You cannot put an except and finally with the same try ...
 7    unless you are running Python 2.5+.
 8 +  The finally clause happens, no matter what, even if there is
 9    a return in the try clause.
10 """
11
12 def PrintFile(f_name):
13     file_obj = open(f_name)
14     try:
15         for line in file_obj:
16             print line,
17     finally:
18         file_obj.close()
19
20 def main(file_name="ram.tzu"):
21     try:
22         PrintFile(file_name)
23     except IOError, msg:
24         print msg
25
26 if __name__ == '__main__':
27     import sys
28     try:
29         main(sys.argv[1])
30     except IndexError:
31         main()
32
33 """
34 $ file2.py
35 Ram Tzu know this:
36 When God wants you to do something,
37 you think it's your idea.
38 $ file2.py xy
39 [Errno 2] No such file or directory: 'xy'
40
41 $"""
```

file3.py

```
#!/usr/bin/env python
"""Demonstrates the 'finally' clause -- and making it happen.
"""

def PrintFile(file_name, fail_on_read=False):
    try:
        open_file = file(file_name)  # file is an alias for open
        try:
            for line in open_file:
                print line,
                if fail_on_read:
                    raise IOError, "Failed while reading."
        finally:
            open_file.close()
    except IOError, msg:
        print msg

def main(file_name="ram.tzu"):
    print """\n    PrintFile("%s")""" % (file_name)
    PrintFile(file_name)
    print """\n    PrintFile("%s", fail_on_read=True)""" % (file_name)
    PrintFile(file_name, fail_on_read=True)
    print """\n    PrintFile("absent_file")"""
    PrintFile("absent_file")

if __name__ == '__main__':
    main()

"""
$ file3.py

    PrintFile("ram.tzu")
Ram Tzu knows this:
When God wants you to do something,
you think it's your idea.

    PrintFile("ram.tzu", fail_on_read=True)
Ram Tzu knows this:
Failed while reading.

    PrintFile("absent_file")
[Errno 2] No such file or directory: 'absent_file'
$
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
45
46 Notes on raising exceptions.  Usually you want to use a
47 built-in exception.
48
49 To see all the built-in exceptions:
50
51 >>> import exceptions
52 >>> help(exceptions)
53
54 or
55
56 >>> help('exceptions')
57
58 """
```

file4.py

```
 1 #!/usr/bin/env python2.5
 2 """The finally can be attached to the try/except since Python 2.5"""
 3
 4 def PrintFile(file_name, fail_on_read=False):
 5     try:
 6         file_obj = open(file_name)
 7         for line in file_obj:
 8             print line,
 9             if fail_on_read:
10                 raise IOError, "Failed while reading."
11     except IOError, msg:
12         print msg
13     finally:
14         try:
15             file_obj.close()
16         except NameError:
17             pass
18
19 def main(file_name="ram.tzu"):
20     print """\n    PrintFile("%s")""" % (file_name)
21     PrintFile(file_name)
22     print """\n    PrintFile("%s", fail_on_read=True)""" % (file_name)
23     PrintFile(file_name, fail_on_read=True)
24     print """\n    PrintFile("absent_file")"""
25     PrintFile("absent_file")
26
27 if __name__ == '__main__':
28     main()
29
30 """
31 $ file4.py > file4.out
32 $ file3.py > file3.out
33 $ diff file3.out file4.out
34 $
35 """
```

Notes about files:

`open` is an alias for `file`:

```
class file(object)
 |  file(name[, mode[, buffering]]) -> file object
 |
 |  Open a file.  The mode can be 'r', 'w' or 'a' for reading (default),
 |  writing or appending.  The file will be created if it doesn't exist
 |  when opened for writing or appending; it will be truncated when
 |  opened for writing.  Add a 'b' to the mode for binary files.
```

(`'b'` is for Macs and Windows – for Unix, `'b'` or no `'b'` is the same)

```
 |  Add a '+' to the mode to allow simultaneous reading and writing.
 |  If the buffering argument is given, 0 means unbuffered, 1 means line
 |  buffered, and larger numbers specify the buffer size.
 |  Note:  open() is an alias for file().
```


Ways to read a file. First:

```
file_object = open('my.txt')
```

then:

```
text = file_object.read()         --> text is a string of the whole file

text = file_object.read(1024)     --> reads 1024 bytes.

lines = file_object.readlines()   --> lines is a list of strings, each a line

one_line = file_object.readline() --> reads one line.

for line in file_object:       ?==?   for line in file_object.readlines():
    print line,                           print line,

^ this form iterates through the      ^ this form puts the whole file in
  file, line by line.                   memory at the same time!
```

You want a comma on your print statement because each line already has a newline.

Ways to write a file. First:

```
file_object = open('my.txt', 'w')   <-- or mode can be 'a' or 'r+'.
```

then:

```
file_object.write(str)
file_object.writelines(list_of_strs)

current_position = file_object.tell()
   gives the read/write head position.

   To move it:

file_object.seek(byte_count, from)
   from = 0 -> beginning of file
        = 1 -> current position
        = 2 -> from end of file

file_object.truncate([byte_count=current_position])

   file_object.seek(0, 0)
   file_object.truncate()

   starts the file over.
```


You can ask:

```
file_object.closed
file_object.mode
file_object.name
```

os Module

Portability issues are packaged into the os module:

```
Linux:                          Windows:
>>> import os                   >>> import os
>>> os.linesep                  >>> os.linesep
'\n'                            '\r\n'
>>> os.sep                      >>> os.sep
'/'                             '\\'
```

Here is a sample of some functions available in `os`. Use `help(os)` for more details:

```
chdir(...)
    Change the current working directory to the specified path.
chmod(...)
    Change the access permissions of a file.
getcwd(...)
    Return a string representing the current working directory.
listdir(...)
    Return a list containing the names of the entries in the
    directory.
mkdir(...)
    Create a directory.
remove(...)
    Remove a file (same as unlink(path)).
rename(...)
    Rename a file or directory.
rmdir(...)
    Remove a directory.
stat(...)
    Perform a stat system call on the given path.
```

os.path:

There are a lot of important functions in the os.path module, which you get for free when you import os.

```
>>> dir(os.path)

['__all__', '__builtins__', '__doc__', '__file__', '__name__',
'_resolve_link', '_varprog', 'abspath', 'altsep', 'basename',
'commonprefix', 'curdir', 'defpath', 'devnull', 'dirname',
'exists', 'expanduser', 'expandvars', 'extsep', 'getatime',
'getctime', 'getmtime', 'getsize', 'isabs', 'isdir', 'isfile',
'islink', 'ismount', 'join', 'lexists', 'normcase', 'normpath',
'os', 'pardir', 'pathsep', 'realpath', 'samefile',
'sameopenfile', 'samestat', 'sep', 'split', 'splitdrive',
'splitext', 'stat', 'supports_unicode_filenames', 'walk']

>>> help(os.path.walk)
Help on function walk in module posixpath:

walk(top, func, arg)
    Directory tree walk with callback function.

    For each directory in the directory tree rooted at top
    (including top itself, but excluding '.' and '..'), call
    func(arg, dirname, fnames). dirname is the name of the
    directory, and fnames a list of the names of the files and
    subdirectories in dirname (excluding '.' and '..').  func
    may modify the fnames list in-place (e.g. via del or slice
    assignment), and walk will only recurse into the
    subdirectories whose names remain in fnames; this can be
    used to implement a filter, or to impose a specific order of
    visiting.  No semantics are defined for, or required of,
    arg, beyond that arg is always passed to func.  It can be
    used, e.g., to pass a filename pattern, or a mutable object
    designed to accumulate statistics.  Passing None for arg is
    common.

>>>
```

walk_.py

```
 1 #!/usr/bin/env python
 2 """walk_.py -- Demonstrates the os.path.walk function, one of many
 3 very useful things given to us in the os module"""
 4
 5 import time
 6 import os
 7
 8 def Func(anything, dirname, fnames):
 9     print anything, os.path.abspath(dirname)
10     for file_name in fnames:
11         whole_name = os.path.join(dirname, file_name)
12         if os.path.isdir(whole_name):
13             print '    directory:', file_name
14         else:
15             print '    %s:\n        %s' % (file_name, time.ctime(
16                 os.path.getmtime(whole_name)))
17
18 if __name__ == '__main__':
19     os.path.walk('cats', Func, 'Walking:')
20
21 """$ walk_.py
22 Walking: /home/marilyn/python/mm/labs/lab_10_File_IO/cats
23     cats.txt:
24         Thu Jun 18 16:21:33 2009
25     more_cats.txt:
26         Thu Jun 18 16:21:33 2009
27     directory: deep_cats
28 Walking: /home/marilyn/python/mm/labs/lab_10_File_IO/cats/deep_cats
29     cats.txt:
30         Thu Jun 18 16:21:33 2009
31     more_cats.txt:
32         Thu Jun 18 16:21:33 2009
33     directory: deeper_cats
34 Walking: /home/marilyn/python/mm/labs/lab_10_File_IO/cats/deep_cats/deeper_cats
35     cats.txt:
36         Thu Jun 18 16:21:33 2009
37     more_cats.txt:
38         Thu Jun 18 16:21:33 2009
39 $
40 """
```

total_text.py

```
#!/usr/bin/env python
"""total_text.py provides a TotalText function, which adds up all the
numbers in the text that it receives."""

import string
punctuation_except_decimal = ''.join(string.punctuation.split('.'))

def TotalText(text, total=0):
    """Returns the sum of all the numbers in the text."""

    words = text.split()
    for word in words:
        word = word.strip(punctuation_except_decimal)
        word = word.rstrip('.')
        try:
            num = float(word)
        except ValueError:
            pass
        else:
            total += num
    return total

def main():
    print "Total =",
    print TotalText("""Here is 1. Add 2 makes 3 or maybe 12,
depending on how you operate.

You might like 2.2 and that's enough unless you like "8.8" or
maybe 1 more or maybe 87. . .5""")

if __name__ == '__main__':
    main()

"""
$ total_text.py
Total = 117.5
$
"""
```

walk_stats.py

```
 1 #!/usr/bin/env python
 2 """Here is a demonstration of using os.path.walk to accumulate
 3 statistics through the files walked."""
 4 import os
 5 import total_text
 6
 7 def TotalDeep(stats, dir_name, files):
 8     """Called by walk to return statistics into stats.
 9     stats = [number_of_files, total]"""
10     print dir_name, "so far: %d files, adding to %d." % (stats[0], stats[1])
11     for file_name in files:
12         pname = os.path.join(dir_name,file_name)
13         if not os.path.isfile(pname):
14             continue
15         try:
16             open_file = open(pname)
17             for line in open_file:
18                 stats[1] += total_text.TotalText(line)
19             open_file.close()
20             stats[0] += 1
21         except IOError, msg:
22             print pname, msg
23
24 def main():
25     """stats is passed into walk for accumulating statistics.  Note that
26     stats must be mutable for this to work."""
27     stats = [0, 0]  # number_of_files, total
28     os.path.walk('cats', TotalDeep, stats)
29     print "That's %d files, totaling to %d." % tuple(stats)
30
31 if __name__ == '__main__':
32     main()
33
34 """
35 $ walk_stats.py
36 cats so far: 0 files, adding to 0.
37 I can't read 'cats/wrong_permissions'.
38 cats/deep_cats so far: 3 files, adding to 12.
39 cats/deep_cats/deeper_cats so far: 5 files, adding to 24.
40 That's 7 files, totaling to 36.
41 $ """
42
```

Lab 10

Collect and extract `labs.zip` from WebCT. From `lab_10_File_IO` you need `do_swap.py`, `cats.txt` and the `cats` directory tree.

`do_swap.py`, printed next, contains a `do_swap.DoSwap(text, apple, orange)` function. Don't look at my `lab10_1.py` and `lab10_3.py` unless you give up, or are short on time.

1. Change (read and rewrite) the file `labs/lab_10_File_IO/cats.txt` so that every *cat* becomes a *dog* and every *dog* becomes a *cat*. Use `do_swap.DoSwap`. Be sure to put your functionality into a function. You'll need it for the next lab.

2. Import your program from the above exercise (or use mine) into a new program that swaps *cat* and *dog* throughout the `cats` directory. Use `os.path.walk`.

   Check that it worked by printing all the files in the directory structure, also using an `os.path.walk`

3. (Optional) Write a program that takes a file path as an argument on the command line, or interactively, and counts how many times each word appears in the text.

   I used the file: `labs/lab_10_File_IO/zen.story` to test.

   Hints:

   - A dictionary is very handy for this. Each word is a `key` into your dictionary; accumulate the count as the `value`.
   - To find the words, you'll have to `split` the lines and `strip` punctuation. `split` and `strip` are builtin string methods.

     The `string` module is mostly obsolete, but does have a `string.punctuation`, a useful string of punctuation characters.

   If you have time, produce a report that lists the 10 most popular words, the count for each word, and a score for each word = (number of times the word appears)/(total words).

   I had a big tie for the 10th most popular word, so I had to list a lot more than 10 words to be fair to each word in the tie.

4. (Optional) Now write a program that depends on the previous exercise to walk a directory structure, accumulating the dictionary of words.

do_swap.py

```
 1 #!/usr/bin/env python
 2 """do_swap.py for importing.
 3
 4 Note that this is a faulty function for swapping occurrences of
 5 'orange' for 'apple' and occurrences of 'apple' for 'orange' in the
 6 text.  It is faulty because it is case-sensitive and because it would
 7 change"category" to "dogegory".  You don't care for this exercise.
 8 We'll get it right when we do regular expressions.
 9 """
10
11 def DoSwap(text, apple, orange):
12     """Swap apple for orange and orange for apple in the text.
13
14     Return the swapped text.
15     """
16
17     dummy = 'wxyz'
18     while True:
19         if text.find(dummy) == -1:
20             break
21         dummy *= 2
22     text = text.replace(apple, dummy)
23     text = text.replace(orange, apple)
24     text = text.replace(dummy, orange)
25     return text
26
27 if __name__ == '__main__':
28     print DoSwap("""Some dogs and cats played together.""", 'dog', 'cat')
29
30 """
31 $ do_swap.py
32 Some cats and dogs played together.
33 $ """
```