Python – S8

Contents

Files

Introduction to Files

- So far we have been reading and writing [data] from/to our standard input and output.
- Now we will try to see how we can use actual data files.
- Python provides basic functions and methods necessary to manipulate files by default.
- You can do most of the file manipulation using a **file** object.

The open function

- Before you can read or write a file, you have to open it using Python's built-in *open()* function.
- This function creates a **file** object, which would be utilized to call other support methods associated with it.
- The general syntax is : file object = open(filename [, access_mode])
- Here access_mode can be ‘r’, ‘w’, ‘a’ – for read, write or append.

Files

Opening Files

- When we want to read or write a file (say on your hard drive), we first must open the file.
- Opening the file communicates with your operating system which knows where the data for each file is stored.
- When you open a file, you are asking the operating system to find the file by name and make sure the file exists.
- In this example, we open the file mbox.txt which should be stored in the same folder that you are in when you start Python.

```
fhand = open('mbox.txt')
>>> print (fhand)
<open file 'mbox.txt', mode 'r' at 0x1005088b0>
```

Files

Opening Files

- If the open is successful, the operating system returns us a file handle.
- The file handle is not the actual data contained in the file, but instead it is a “handle” that we can use to read the data.
- You are given a handle if the requested file exists and you have the proper permissions to read the file.

- If the file does not exist, open will fail with a traceback and you will not get a handle to access the contents of the file:

```
>>> fhand = open('stuff.txt')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'stuff.txt'
```

Files

Opening Files

- A text file can be thought of as a sequence of lines, much like a Python string can be thought of as a sequence of characters.
- To break the file into lines, there is a special character that represents the “end of the line” called the newline character.
- In Python, we represent the newline character as a backslash-n in string constants.
- Even though this looks like two characters, it is actually a single character.

Files

Reading Files

- While the file handle does not contain the data for the file, it is quite easy to construct a for loop to read through and count each of the lines in a file:

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

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

- We can use the file handle as the sequence in our for loop. Our for loop simply counts the number of lines in the file and prints them out. The rough translation of the for loop into English is, “for each line in the file represented by the file handle, add one to the count variable.”

Files

Reading Files

- When the file is read using a for loop in this manner, Python takes care of splitting the data in the file into separate lines using the newline character.
- Python reads each line through the newline and includes the newline as the last character in the line variable for each iteration of the for loop.
- Because the for loop reads the data one line at a time, it can efficiently read and count the lines in very large files without running out of main memory to store the data.
- The above program can count the lines in any size file using very little memory since each line is read, counted, and then discarded.
- If you know the file is relatively small, you can read the whole file into one string using the read method on the file handle.

Files

Reading Files

```
>>> fhand = open('mbox-short.txt')
>>> inp = fhand.read()
>>> print (len(inp))
94626
```

- In this example, the entire contents (all 94,626 characters) of the file ‘mbox-short.txt’ are read directly into the variable inp.
- When the file is read in this manner, all the characters including all of the lines and newline characters are one big string in the variable inp.
- Remember that this form of the open function should only be used if the file data will fit comfortably in the main memory of your computer.
- If the file is too large to fit in main memory, you should write your program to read the file in chunks using a for or while loop.

Files

Searching through a File

- When you are searching through data in a file, it is a very common pattern to read through a file, ignoring most of the lines and only processing lines which meet a particular criteria.
- We can combine the pattern for reading a file with string methods to build simple search mechanisms.

- For example, if we wanted to read a file and only print out lines which started with the prefix “From:”, we could use the string method ‘startswith’ to select only those lines with the desired prefix:

Files

Searching through a File

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

- When this program runs, we get the following output:

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

- The output will however also print some blank lines, as they are generated because of newline character.

Files

Searching through a File

- So if you want to remove them, just modify the program as :

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

Files

Searching through a File

- As your file processing programs get more complicated, you may want to structure your search loops using continue.
- The basic idea of the search loop is that you are looking for “interesting” lines and effectively skipping “uninteresting” lines.
- And then when we find an interesting line, we do something with that line.
- We can structure the loop to follow the pattern of skipping uninteresting lines as follows:

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    # Skip 'uninteresting lines'
    if not line.startswith('From:') :
        continue
    # Process our 'interesting' line
    print (line)
```

Files

Searching through a File

- The output of the program is the same.
- In English, the uninteresting lines are those which do not start with “From:”, which we skip using continue.
- For the “interesting” lines (i.e. those that start with “From:”) we perform the processing on those lines.
- We can use the find string method to simulate a text editor search which finds lines where the search string is anywhere in the line.
- Since find looks for an occurrence of a string within another string and either returns the position of the string or -1 if the string was not found, we can write the following loop to show lines which contain the string “@uct.ac.za”

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if line.find('@uct.ac.za') == -1 :
        continue
    print (line)
```

Files

Letting the user choose the filename

- We really do not want to have to edit our Python code every time we want to process a different file.
- It would be more usable to ask the user to enter the file name string each time the program runs so they can use our program on different files without changing the Python code.
- This is quite simple to do by reading the file name from the user using 'input' as follows:

```
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)
```

Files

Using try, except and open

- We stated earlier that if we supply a filename which does not exists, we will get an error and program stops there.
- In order to elegantly take care of the problem, we should use the try/ except structure, as shown below :

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

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

Files

Writing Files

- To write a file, you have to open it with mode 'w' as a second parameter:

```
>>> fout = open('output.txt', 'w')
```

- If the file already exists, opening it in write mode clears out the old data and starts fresh, so be careful!

- If the file doesn't exist, a new one is created.
- The write method of the file handle object puts data into the file.

```
>>> line1 = 'This here's the wattle,\n'
>>> fout.write(line1)
```

- Again, the file object keeps track of where it is, so if you call write again, it adds the new data to the end.

Files

Writing Files

- We must make sure to manage the ends of lines as we write to the file by explicitly inserting the newline character when we want to end a line.
- The print statement automatically appends a newline, but the write method does not add the newline automatically.

```
>>> line2 = 'the emblem of our land.\n'
>>> fout.write(line2)
```

- When you are done writing, you have to close the file to make sure that the last bit of data is physically written to the disk so it will not be lost if the power goes off.

```
>>> fout.close()
```

- We could close the files which we open for read as well, but we can be a little sloppy if we are only opening a few files since Python makes sure that all open files are closed when the program ends.
- When we are writing files, we want to explicitly close the files so as to leave nothing to chance.

Renaming and Deleting Files

- Python ‘os’ module provides methods that help you perform file-processing operations, such as renaming and deleting files.
- To use this module you need to import it first and then you can call any related functions.
- The *rename()* method takes two arguments, the current filename and the new filename.
- The syntax is : os.rename(current_file_name, new_file_name)

```
import os
os.rename(‘test1.txt’, ‘test2.txt’)
```

- You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.

```
import os
os.remove(“test2.txt”)
```

A little more into Files

Reading an Entire File

- To begin, we need a file with a few lines of text in it. Let's start with a file that contains *pi* to 30 decimal places with 10 decimal places per line:

3.1415926535

8979323846

2643383279

- To try the following examples yourself, you can enter these lines in an editor and save the file as *pi_digits.txt*. Save the file in the same directory where your programs are saved.
- Here's a program that opens this file, reads it, and prints the contents of the file to the screen:

```
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

- The open() function needs one argument: the name of the file you want to open.
- Python looks for this file in the directory where the program that's currently being executed is stored.
- The open() function returns an object representing the file.

Reading an Entire File

- Here, open('pi_digits.txt') returns an object representing *pi_digits.txt*.
- Python stores this object in file_object, which we'll work with later in the program.
- The keyword 'with' closes the file once access to it is no longer needed.
- Notice how we call open() in this program but not close().
- You could open and close the file by calling open() and close(), but if a bug in your program prevents the close() statement from being executed, the file may never close.
- This may seem trivial, but improperly closed files can cause data to be lost or corrupted.
- It's not always easy to know exactly when you should close a file, but with the structure shown here, Python will figure that out for you.
- All you have to do is open the file and work with it as desired, trusting that Python will close it automatically when the time is right.
- The rest of the program is quite easy to follow.
- Please note : You can supply the entire filename path in the argument of 'open'.

with open('/home/prasun/text_files/*filename*.txt') as file_object:

Reading Line by Line

- You can use a for loop on the file object to examine each line from a file one at a time:

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    for line in file_object:
        print(line)
```

- The output of this program will have blank lines between each lines, because an invisible ‘newline’ character is at the end of each line in the text file.
- The print statement adds up its own newline each time we call it!!!
- So there is 2 newlines at end of each line.

- To remove those, you can use :

```
print(line.rstrip())
```

Making a List of Lines from a File

- The following example stores the lines of *pi_digits.txt* in a list inside the with block and then prints the lines outside the with block:

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()

for line in lines:
    print(line.rstrip())
```

- The readlines() method takes each line from the file and stores it in a list.
- After you've read a file into memory, you can do whatever you want with that data.
- For example :

Making a List of Lines from a File

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
        lines = file_object.readlines()
pi_string = ''
for line in lines:
        pi_string += line.rstrip()
print(pi_string)
print(len(pi_string))
```

- The output will be :

```
3.1415926535  8979323846 2643383279
36
```

- To remove the blank spaces from the lines which existed in the text file, you can use :

```
pi_string += line.strip()
```

- Now the output becomes:

```
3.141592653589793238462643383279
32
```

Writing to a File

- One of the simplest ways to save data is to write it to a file.
- When you write text to a file, the output will still be available after you close the terminal containing your program's output.
- You can examine output after a program finishes running, and you can share the output files with others as well.
- You can also write programs that read the text back into memory and work with it again later.
- To write text to a file, you need to call open() with a second argument telling Python that you want to write to the file.
- To see how this works, let's write a simple message and store it in a file instead of printing it to the screen:

```
filename = 'programming.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
```

- You can open a file in *read mode* ('r'), *write mode* ('w'), *append mode* ('a'), or a mode that allows you to read and write to the file ('r+').
- If you omit the mode argument, Python opens the file in read-only mode by default.

Writing Multiple Lines

- The write() function doesn't add any newlines to the text you write.
- So if you write more than one line without including newline characters, your file may not look the way you want it to.
- Including newlines in your write() statements makes each string appear on its own line:

```
filename = 'programming.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.\n")
    file_object.write("I love creating new games.\n")
```

- The output now appears on separate lines:

```
I love programming.
I love creating new games.
```

Appending to a File

- If you want to add content to a file instead of writing over existing content, you can open the file in *append mode*.
- When you open a file in append mode, Python doesn't erase the file before returning the file object.
- Any lines you write to the file will be added at the end of the file.
- If the file doesn't exist yet, Python will create an empty file for you.

```
filename = 'programming.txt'
with open(filename, 'a') as file_object:
    file_object.write("I also love finding meaning in large datasets.\n")
    file_object.write("I love creating apps that can run in a browser.\n")
```

- The content of the file written now becomes :

I love programming.

I love creating new games.

I also love finding meaning in large datasets.

I love creating apps that can run in a browser.

Files - Example programs

Example program for Files

```
# to copy a file to another, but adding a line no for each line


f_in = open (“abc.txt”)
f_out = open(“abc2.txt”, ‘w’)
i = 1
for line in f_in:
        f_out.write(str(i) + “: “ + line)
        i = i + 1
f_in.close()
f_out.close()
```

Files - Example programs

```
# In this example we will copy a given text file to another file. copyfile.py
import sys
if len(sys.argv) < 3:
        print("Wrong parameter")
        print("./copyfile.py file1 file2”)
        sys.exit(1)
f1 = open(sys.argv[1])
s = f1.read()
f1.close()
f2 = open(sys.argv[2], 'w')
f2.write(s)
f2.close()
```

Note : This way of reading file is not always a good idea, a file can be very large to read and fit in the memory. It is always better to read a known size of the file and write that to the new file.

Files

Another Example program for Files

- Say we have a text file like below, called 'energy.log':

Starting program...

Loading molecule...

Initialising variables...

Starting the calculation - this could take a while!

Molecule energy = 2432.6 kcal mol-1

Calculation finished. Bye!

- How to know the energy value by searching lines that contain "Molecule energy ="

```
import re
lines = open ("energy.log", "r").readlines()
for line in lines:
    if re.search(r"Molecule energy =", line) :
        words = line.split()
        energy = float( words[3] )
        print("The energy of the molecule is %f kcal mol-1" % energy)
        break
```

Files

Another Example program for Files

```
## Create, manage Account Master file
def add_rec():
    rec = ''
    acctype = str(input("Account Type (SB/CA/FD) : "))
    roi = str(input("Rate of Interest : "))

    rec = acctype + ',' + roi + '\n'

    with open('account_master.txt', 'a') as file:
       file.write(rec)

def show_all_rec():
    with open('account_master.txt', 'r') as file:
       for line in file:
          print(line)
```

[contd.]

Files

Another Example program for Files [contd.]

```
print('============================\n')
print('Account Master Maintenance \n')
print('============================\n')
print(' Please choose from the following option : ')
print()
print('1. To ADD a record \n' )
print('2. To SHOW all records \n')

choice = int(input("Please Enter your option : "))
if choice == 1:
   add_rec()
elif choice == 2:
   show_all_rec()
else:
   print("Invalid choice ....!!!!\n")
```

Files

Count spaces, tabs and new lines in a file [ex14.py]

```
import os
import sys
def parse_file(path):
    """
    Parses the text file in the given path and returns space, tab & new line details.
    :arg path: Path of the text file to parse
    :return: A tuple with count of spacaes, tabs and lines.
    """
    fd = open(path)
    i = 0
    spaces = 0
    tabs = 0
    for i,line in enumerate(fd):
        spaces += line.count(' ')
        tabs += line.count('\t')
    #Now close the open file
    fd.close()
    #Return the result as a tuple
    return spaces, tabs, i + 1
```

contd…

Files

Count spaces, tabs and new lines in a file [contd from previous slide]

```
def main(path):
    """
    Function which prints counts of spaces, tabs and lines in a file.
    :arg path: Path of the text file to parse
    :return: True if the file exits or False.
    """
    if os.path.exists(path):
        spaces, tabs, lines = parse_file(path)
        print("Spaces %d. tabs %d. lines %d" % (spaces, tabs, lines))
        return True
    else:
        return False


if __name__ == '__main__':
    if len(sys.argv) > 1:
        main(sys.argv[1])
    else:
        sys.exit(-1)
    sys.exit(0)
```

Files

Count spaces, tabs and new lines in a file [contd from previous slide]

The usage will be :

$ ex14.py anytextfile

The output will be something like :

$ Spaces 278. tabs 0. lines 49

Files

To remove comments in a python source file

Program Name : to-remove-comments.py

Show the program

To run

$ python to-remove-comments.py

Files

To check for duplicate files from a given path

Program Name : to-find-duplicate-files.py

Show the program

To run

$ python to-find-duplicate-files.py <path>

Reference

- Python for Informatics – C. Severance
- Think Python – A. B. Downey - [O'Reilly]
- Python Crash Course – Eric Matthes [No starch Press]
- A Byte of Python – Swaroop C H
- Introducing Python – Bill Lubanovic [O'Reilly]

- Learning Python – 5th Ed – Mark Lutz - [O'Reilly]

End of Presentation

Python – S8