Python: Files (Part I)

9th Lecture

1. Introduction

Thus far in this lectures, we have seen examples of programs that have taken input data *from users at the keyboard*. Most of these programs can receive their input from ***files*** as well. A **file** ***is a software object that stores data on a permanent medium such as a disk, CD, or flash memory***. When compared to keyboard input from a human user, the main advantages of taking input data from a file are the following:

- The data set can be much larger.
- The data can be input much more quickly and with less chance of error.
- The data can be used repeatedly with the same program or with different programs.

2. Files and Their Format

Using a text editor such as Notepad or TextEdit, you can create, view, and save data in a file. Your Python programs can output data to a text file, a procedure explained later in this section. The data in a file can be viewed as characters, words, numbers, or lines of text, depending on the file's format and on the purposes for which the data are used. When the data are treated as numbers (either integers or floating-points), they must be separated by white- space characters—spaces, tabs, and newlines. *For example*, a text file containing six floating-point numbers might look like:

```
34.6 22.33 66.75
77.12 21.44 99.01
```

When examined with a text editor. Note that this format includes a space or a newline as a separator of items in the text.

All data output to or input from a file must be strings. Thus, numbers must be converted to strings before output, and these strings must be converted back to numbers after input.

➢ Opening a File

Before reading/writing you first need to open the file. Syntax of opening a file is. The syntax of File as follows:

fileNamePointer = open("*filename* ", "*mode*")

open() function accepts two arguments **filename** and **mode** . ***filename*** is a string argument which specify filename along with its path and ***mode*** is also a string argument which is used to specify how file will be used *i.e.,* for reading or writing. And ***fileNamePointer*** is a file handler object also known as file pointer.

Different modes of opening a file are:

Modes	Description
"r"	Open a file for read only
"w"	Open a file for writing. If file already exists its data will be cleared before opening. Otherwise new file will be created
"a"	Opens a file in append mode i.e. to write a data to the end of the file
"wb"	Open a file to write in binary mode
"rb"	Open a file to read in binary mode

➢ Closing a File

After you have finished reading/writing to the file you need to close the file using *close()* method like this,

fileNamePointer.close() *# where fileNamePointer is a file pointer*

3. Writing to a File

Data can be output to a file using a file object. Python's open function, which expects a file pathname and a mode string as arguments, opens a connection to the file on disk and returns a file object. The mode string **'w'** for output files. Thus, the following code opens a file object on a file named ***myfile.txt*** for output. The syntax of write to a file as follows:

fileNamePointer = open("*myfile.txt*", '*w*')

If the file does not exist, it is created with the given pathname. If the file already exists, Python opens it. When data are written to the file and the file is closed, any data previously existing in the file is erased. String data are written (or output) to a file using the method write with the file object. The write method expects a single string argument. If you want the output text to end with a newline, you ***must include the escape character \n*** in the string. The next statement writes one line of text to the file:

fileNamePointer.write("Write lines.\n")

When all of the outputs are finished, the file should be closed using the method close, as follows:

fileNamePointer.close()

Example 1:

```
f = open('myfile.txt', 'w')          # open file for writing
f.write('this first line\n')          # write a line to the file
f.write('this second line\n')         # write one more line to the file
f.close()                             # close the file
```

Example 2: Writing numbers to a file?

```
import random
f = open("integers.txt", 'w')
for count in range(500):
    number = random.randint(1, 500)
    f.write(str(number) + "\n")
f.close()
```

4. Reading from a File

You open a file for input in a manner similar to opening a file for output. The only thing that changes is the mode string, which, in case of opening a file for input, is '**r**'. However, if the pathname is not accessible from the current working directory, Python raises an error. Here is the code for opening myfile.txt for input:

fileNamePointer = open("myfile.txt", 'r')

There are several ways to read data from an input file. The simplest way is to use the file method read to input the entire contents of the file as a single string. If the file contains multiple lines of text, the newline characters will be embedded in this string. To read data back from the file you need one of these three methods:

```
read()        #Read all the lines as a list of strings in the file
readline()    #Return the next line of the file.
readlines()   #Read all the lines as a list of strings in the file
```

The next example shows how to use the method read:

Example 3:

```
f = open('myfile.txt', 'w')          # open file for writing
f.write('this first line\n')          # write a line to the file
f.write('this second line\n')         # write one more line to the file
f.close()                             # close the file

f = open('myfile.txt', 'r')           # open file for reading
text = f.read()
print(text)
```

The output
this first line
this second line

Note: *In this case, it is not necessary to close the file.*

To *repeat an input*, the file must be ***re-opened***. Alternatively, an application might read and process the text one line at a time. A ***for*** loop accomplishes this nicely. The ***for*** loop views a file object as a sequence of lines of text. On each pass through the loop, the loop variable is bound to the next line of text in the sequence. Here is a session that re-opens our example file and visits the lines of text in it:

Example 4:

```
f = open('myfile.txt', 'w')          # open file for writing
f.write('this first line\n')          # write a line to the file
f.write('this second line\n')         # write one more line to the file
f.close()                             # close the file


f = open('myfile.txt', 'r')           # open file for reading
while True:
    line = f.readline()
    if line == "":
        break
    print(line)
```

The output
this first line
this second line

Example 5: Reading numbers from a file. When reading data from a file, another important consideration is the format of the data items in the file. Earlier, we showed an example code segment that output integers **separated** by newlines to a text file. During input, these data can be read with a simple **for** loop. This loop accesses a line of text on each pass. To convert this line to the integer contained in it, the programmer runs the **string** method ***strip*** to remove the newline and then runs the ***int*** function to obtain the integer value. The next code segment illustrates this technique. It opens the file of random integers written earlier, reads them, and prints their sum.

```
import random
f = open("integers.txt", 'w')
for count in range(500):
   number = random.randint(1, 500)
   f.write(str(number) + "\n")
f.close()

f = open("integers.txt", 'r')
sum = 0
for line in f:
    line = line.strip()
    number = int(line)
    sum += number
print("The sum is", sum)
```

The output
The sum is 121956

Obtaining numbers from a file in which they are separated by spaces is a bit trickier. One method proceeds by reading lines in a ***for*** loop, as before. But each line now can

contain several integers *separated* by spaces. You can use the string method ***split*** to obtain a list of the strings representing these integers, and then process each string in this list with another for loop. The next code segment modifies the previous one to handle integers separated by spaces and/or newlines.

```
import random
f = open("integers.txt", 'w')
for count in range(500):
   number = random.randint(1, 500)
   f.write(str(number) + "\n")
f.close()

f = open("integers.txt", 'r')
sum = 0
for line in f:
   wordlist = line.split()
   for word in wordlist:
      number = int(word)
       sum += number
print("The sum is", sum)
```

The output
The sum is 121956

5. Appending Data to a File

To append the data you need to open the file in **'a'** mode.

Example :

```
f = open('myfile.txt', 'a')          # open file for appending
f.write("this is third line\n")
f.close()
```

6. Exercises

1. Write a code segment that opens a file for input and prints the number of four-letter words in the file.
2. Assume that a file contains integers separated by newlines. Write a code segment that opens the file and prints the average value of the integers.
3. Write a Python program to read a file line by line and store it into a list

<Best Regards>

Dr. Raaid Alubady