**7.2 Text Files**

So far, our codes have had input from the user, and the output has been displayed. It is also possible to read data from a file, and it is possible to write results to a file.

When working with files, there are three basic steps:

1. Open the file
2. Use the file
3. Close the file

A file can be opened to read from it, write to it (which means from the beginning of the file), or ***append*** to it (which means writing to it from the end; adding to an existing file). Writing, reading, and appending are called ***modes***. The function that opens a file is called **open**. The general form of calling it is:

*>>> f = open(“filename”, mode)*

where the file to be opened is in the correct folder so that the open function can locate it. The mode is indicated by a string:

* ‘r’ for reading (which is the default)
* ‘w’ for writing
* ‘a’ for appending
* ‘r+’ to both read from and write to a file

Files that have been opened and used must be closed using the **close** method:

*>>> f.close()*

**7.2.1 Writing to a file**

To write to a file, there are two options: writing to a file, and appending to a file. If the file is opened for writing using the mode ‘w’, lines are written to the file from the beginning. If the file already existed, the contents of the file are erased. If the mode ‘a’ is specified, that assumes that the file already exists, and the writing will begin at the end of the file. There are two main methods that can be used:

* **write(s)** to write a string *s* to the file
* **writelines(ls)** to write a list *ls* to the file

The **writelines** method writes an entire list of strings to a file, one string per line in the file.

The more practical method to use is **write**, which writes one string to a file. Since the **write** method only writes one string to a file, it is generally in a loop (either a **while** loop or a **for** loop). Note that the newline character will usually be at the end of each string, so that every string is written to a separate line in the file. For example, the following opens a file for writing, creates a file with 3 lines in it, and then closes the file.

f = open("tryit.txt", 'w')

for i in range(3):

f.write("Hello\n")

f.close()

**7.2.2 Reading from a file**

To read from a file once it has been opened, there are three main methods that can be used:

* **read** to read in the entire file as one string
* **readlines** to read all of the lines from the file (into a list of strings)
* **readline** to read one line from the file as a string

Since files can be very large, it is generally impractical to use **read** or **readlines**. The **readline** method is normally used. However, with the short file we created in the previous section, we will demonstrate all of these methods. The **read** method reads in all characters, including the newline characters into a string. Below we print just the first 8 characters in the string, which includes a ‘\n’. (Note: the newline character ‘\n’ is a single character).

f = open("tryit.txt")

onestr = f.read()

print(onestr[0:8])

f.close()

Hello

He

The **read** method returns an empty string if the end of file has been reached.

f = open("tryit.txt")

onestr = f.read()

print(onestr)

onestr = f.read()

print(repr(onestr))

Hello

Hello

Hello

''

The **readlines** method reads all lines from the file into a list of strings.

f = open("tryit.txt")

allstrs = f.readlines()

allstrs

['Hello\n', 'Hello\n', 'Hello\n']

The **readlines** method returns an empty list if the end of file has been reached.

The **readline** method reads one line at a time into a string. Since we know that our file has 3 lines in it, we will loop to use readline 4 times.

f = open("tryit.txt")

for i in range(4):

oneline = f.readline()

print(repr(oneline))

'Hello\n'

'Hello\n'

'Hello\n'

''

As can be seen from the output, the **readline** method returns an empty string if the end of file has been reached.

In general, when reading from a file, the procedure is to read one line at a time and process it until the end of the file has been reached. The general form that accomplishes this using a **while** loop is:

*f = open(“filename”)*

*aline = f.readline()*

*while aline:*

*# do something with the line*

*aline = f.readline()*

*f.close()*

Note that the mode ‘r’ does not need to be specified since reading is the default. Also, since an empty string is a way of representing **False**, this loops until the **readline** method returns an empty string, meaning that the end of the file has been reached.

In Python, this can also be accomplished using a **for** loop that iterates through the lines in a file object.

*f = open(“filename”)*

*for aline in f:*

*# do something with the line*

*f.close()*

Whether reading or writing, files that have been opened should always be closed after the file processing has been completed. It is easy to forget to do that, however! An alternative is to use the **with** statement, which automatically closes the file. The general form is:

*with open(filename) as f:*

*# code to process file*

The following example uses **with** for both writing to a file and reading from a file.

*with open("greets.txt", 'w') as f:*

*f.write("Hello\n")*

*f.write("Hi\n")*

*f.write("Ciao\n")*

*with open("greets.txt") as f:*

*for aline in f:*

*print(aline.rstrip())*

Hello

Hi

Ciao

The **rstrip** method was used to strip the ‘\n’ from the end of each string before printing it; otherwise, there would be blank lines between each in the output.