Chapter 8: Files and I/O

Up to this point, all input and output has taken place using the keyboard and the command console. Specifically, the function `input()` has been used to collect user data, and the function `print()` has been used to offer information and output to the user.

However, it is very common to collect data and information from a file (or database). Similarly, it is common to write output to a file, or another permanent location, rather than to a command console. Chapter 8 will focus on file utilization, including creating files, reading files, writing and appending to files, and other related topics.

Exercise 8.1: The Spyder File Explorer

As a first and very important step, make sure you can locate the Python programs you have been creating throughout this book. As noted, this book uses the Spyder IDE, which offers a **File Explorer** view on the upper right of the development environment. Most IDEs offer a file explorer option.

From a command console, type `ls` to get a list of the local files. This way, when you create a new file to write to and read from, you will know its location. Also assure that you can locate this case file and folder location from your computer.

Example: Type ls to the console

```
ls


 Directory of C:\Users\Ami\Documents\Python Scripts

07/30/2016  09:50 PM    <DIR>          .
07/30/2016  09:50 PM    <DIR>          ..
06/16/2016  09:32 PM    <DIR>          __pycache__
06/22/2016  10:03 PM                86 A_new_File.txt
06/16/2016  08:59 PM               125 AddNumbers.py
07/17/2016  09:40 PM             3,596 AllGraphsChapter9.py
06/22/2016  10:25 PM                10 AmiFile1.txt
07/30/2016  09:57 PM               756 AverageByIndex.py
…
```

The following set of examples will illustrate the opening (creation) of a new file, adding text to the file, printing the contents of the file, and closing the file. It will also display the location of the file in the File Explorer.

8.1.1: Creating a new file

To create a new file in Python, the **open** function will be used. The syntax is:

```
MyFile = open(<file name>, "w")
```

The "w" stands for "write". If the file was already created, it will be overwritten. Other open options are "a" for append (which will begin writing to the file at the next available location), "r" for reading, and "r+" for reading and writing.

Example 8.1.1: Creating or Opening a File

The **open** function in Python will create a new file, or will open an existing file. This **mode** will determine how the file will be used.

The syntax for the open function is:

```
MyFile=open(Filename, mode)
```

The `Filename` can be any string that is the name of a file. The extension of the file, such as .doc, .docx, .xlsx, or .txt should be included. The `mode` can be "w" for write, "a" for append, "r+" for reading and writing, "r" for reading only. The mode parameter is optional and will default to "r" if not included.

If a file does not exist, it cannot be opened for reading. For example, the file MyFile1.txt does not yet exist. Therefore, calling `FileHandle=open("MyFile1.txt", "r")` will cause the following error:

Example of error for calling open on a file that does not exist:

```
FileHandle=open("MyFile1.txt", "r")
Traceback (most recent call last):

  File "<ipython-input-3-478779897f23>", line 1, in
<module>
    FileHandle=open("MyFile1.txt", "r")

FileNotFoundError: [Errno 2] No such file or directory:
'MyFile1.txt'
```

To create a new file, the mode "w" or "a" must be used. The "w" mode will create a file for writing, and the "a" mode will create a file for appending. If a file does exist, opening it with the "w" (write mode) will erase all file contents. Essentially, opening a file that does exist with the "w" mode will **write over** the file contents.

To write to a file that already exists, open the file with the “a” (append) mode. The “r” or read mode can be used to open a file that already exists, for reading only. The “r+” mode can be used to open a file that already exists for reading or writing.

Exercise 8.1.1: Creating and locating a new file

To complete this exercise, follow these steps:

1) Create a new called, MyNewFile.txt using this line of code:

```
file1=open("MyNewFile.txt", "w")
```

2) Locate this new file inside the Spyder File explorer. Figure 8.1 will illustrate this step.

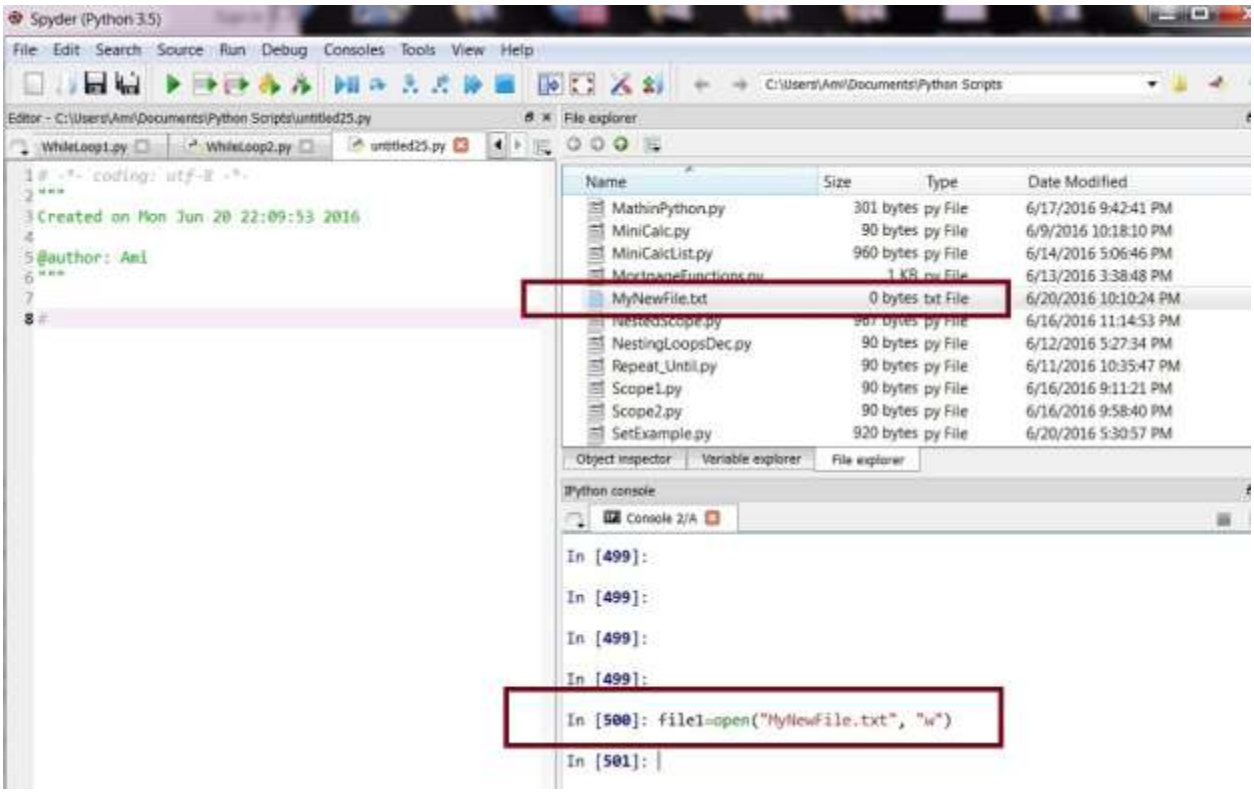

Figure 8.1: Creating a new file in Python

In the above image example, the statement: `file1=open("MyNewFile.txt", "w")` was used to create a new file for writing (*"w"*). The name of the new file is *"MyNewFile.txt"*. Once *"MyNewFile.txt"* is created, it will become visible in the **File Explorer**. In addition, the **file handle**, named `file1` will allow access to reading from, writing to, closing, appending, or deleting that file. A file handle can be thought of as a variable that points to or manages the file.

8.1.2: Writing to a file

When a file is created in Python, a file handle (or name associated with the file management) can be part of the file creation. From the above subsection, the file "MyNewFile.txt" was created, and the file handle that manages that file is called, `file1`.

The syntax for writing to a file is:

```
FileHandle.write(<string>)
```

Example:

```
file1.write("This will be written to the first line of the
file\n")
Out[42]: 51
```

The output of the write method is the number of characters (51 in this case) that have been written to the file. The "\n" is a **newline** symbol that forces the file pointer to go to the next line of the file. This is like pressing the "enter" key after typing.

Exercise 8.1.2: Writing to a File and Confirming the Contents

This exercise will illustrate the steps of opening a file for writing, writing to that file, and then closing the file. Next, the exercise will illustrate how to locate and confirm the contents of the new file in the Spyder File explorer area.

Follow these steps:

1) Create a new file called MyNewFile.txt. If the file already exists, open it for writing.

```
file1=open("MyNewFile.txt", "w")
```

2) Visually confirm that the new file is in the File explorer.

3) Write the sentence to the file: *"This will be written to the first line of the file\n "*

```
file1.write("This will be written to the first line of the
file\n")
```

4) Close the file:

```
file1.close()
```

5) Locate and open the file in the File explorer. You should see the sentence inside the file. Figure 8.2 will illustrate these steps.

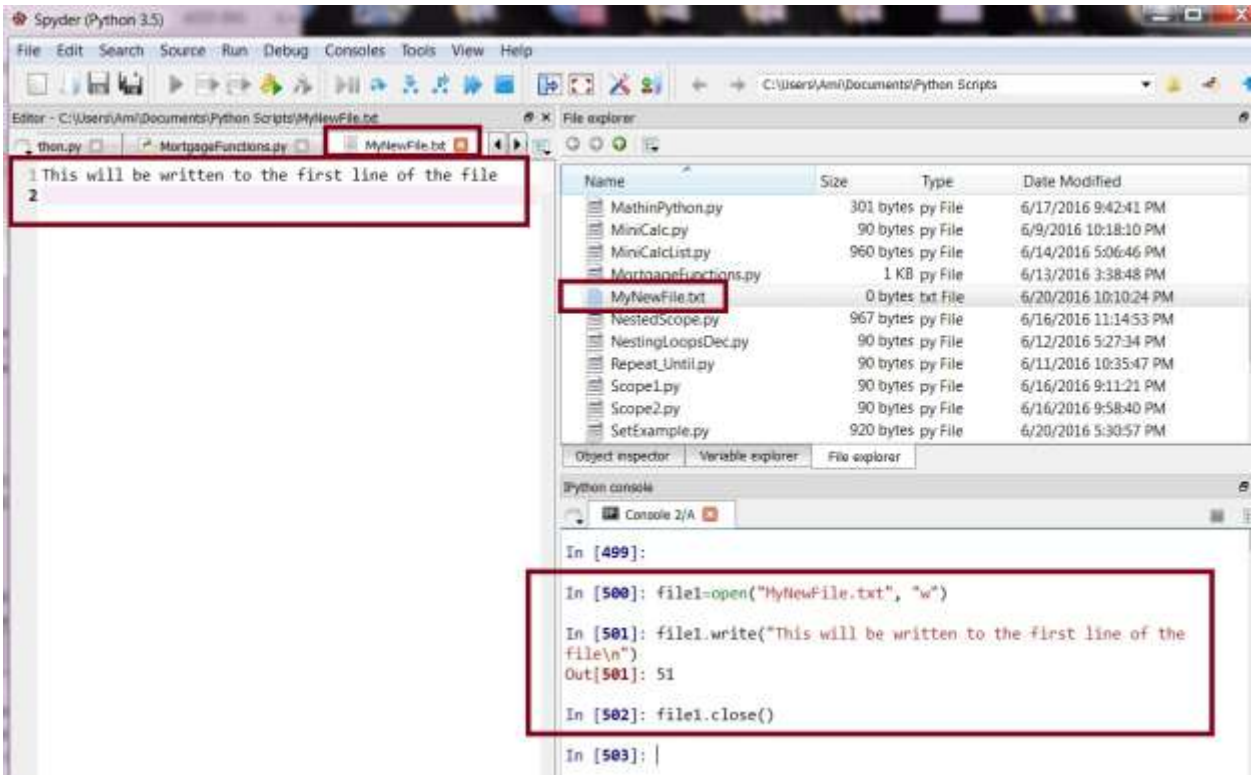

Figure 8.2: Contents of a new file in Spyder

8.1.3: Closing a File

It is imperative that all files not being used in Python are properly closed. Leaving a file open can create any number of issues. In addition, if a file is open for writing ("w"), it must be closed and then re-opened if the contents are to be read ("r").

The syntax for closing a file is:

```
FileHandle.close()
```

In this next example, a file will be opened for reading and writing. A few lines will be written to the file, and the contents of the file will be read and saved in a variable.

8.1.4: Reading a File

Reading the contents of a file can be accomplished using several methods. This section will cover the `readline()` method, the `read()` method, and the use of a `for/in` loop for reading.

When a file is read, the contents of the file can be stored in a variable or structure and can be written or printed to another location. Reading a file does not alter or disturb the contents of the

file. For the following syntax, assume that a file has already been created and that `FileHandle` manages that file.

The syntax for the **readline** function follows. The value `n` is optional. If `n` is omitted, the readline function will read the next entire line of the file. If included, the `n` option will specify the number of characters to be read. For example, if `n` is 5, exactly 5 characters will be read from that line in the file.

The readline function is intended to read one line of the file and then to stop reading when the end of the line (the **newline** character, "\n") is reached. The newline character is generally hidden in a file (not visible when the file is opened). It can be viewed by calling the read function, or by noting that when a new line of the file occurs, a "\n" must have occurred on the previous line. The \n newline is like "Enter" or "Return" on a keyboard. Once read, the line in the file can be printed to output, can be saved into a variable, and can also be written to another file.

```
FileHandle.readline(n)

print(FileHandle.readline(n))
LineOfFile = FileHandle.readline(n)
```

The syntax for the **read** function is:

```
FileHandle.read(n)

print(FileHandle.read(n))
FileContents = FileHandle.read(n)
```

The read function is similar to the readline function in that the n is optional and specifies the number of characters to be read. However, the readline function will only read the current line and not more (if n exceeds the length of the line, readline will still stop reading once the line ends). However, the read function will read the entire file at once. The read function can save all contents of the file to a variable and can also write or print the contents elsewhere.

The syntax for using a for/in loop to read a file is:

```
for line in FileHandle:
     print(line)
```

By using the for/in loop, each line of the file is placed into the variable called, `line`, which can then be printed or manipulated as desired.

8.1.5: File Handles, seek(), and tell()

File Handles

When a new file is created or opened, it is associated with a variable, often called a **file handle** or **file pointer** that manages operations on the file. The file handle is used to read from the file, to write to the file, and to close the file.

For example, the statement:

```
file1=open("MyNewFile.txt", "w")
```

opens (or creates if not yet created) the file called *MyNewFile.txt* and uses the variable (file handle) called, `file1` to manage operations on the file.

As data or information is written into a file or read from a file, the file handle keeps track of the current location in the file. In other words, it **points** to the current location in the file at any given moment. When a file is first created or opened, the file handle variable points to the start of the file, also known as location "0".

As information is written to the file, the file handle moves through the file so it can continue to write new information into the next location and not over existing information. The best way to imagine this is to think about the cursor that is seen as you type into any document. As you type, the cursor moves along with you, adding characters as you go. You have the option to relocate and control the cursor location.

The file handle is very much like a cursor or pointer, and it can be controlled and relocated with two functions, `seek()` and `tell()`.

The seek() function

Within a file, the current location of the file handle can be updated (altered) by calling the `seek()` function. For the next several examples, assume that a file has already been created and has been opened using the following statement, which allows the file to be read from or written to:

```
FileHandle=open("MyNewFile.txt", "r+")
```

The following syntax illustrates the seek function:

```
FileHandle.seek(n)
```

The seek function requires one parameter, called n. The variable n is the location in the file (assuming that the first character on the file is position 0). For example, `FileHandle.seek(0)` will move the file handle to the beginning of the file (position 0).

Similarly, `FileHandle.seek(25)` will move the file handle to the position of the 25th character.

The tell() function

Within a file, the current location of the file handle can be determined by calling the `tell()` function.

The following syntax illustrates the tell function:

```
Location = FileHandle.tell()
```

The tell function requires no parameters. It will return the current location of the file handle. For example, if the file handle is moved to location "10" in the file using the seek function, then the tell function will return a "10".

```
FileHandle.seek(10)

FileHandle.tell()
Out[84]: 10
```

Together, `seek()` and `tell()` can be used to move around a file and read and write as desired. The following program will illustrate several of these file manipulation concepts.

```
# Data_I_O_example.py
# Ami Gates
# Create a new file for writing
fpointer1=open("DataFile.txt", "w")
# Get data from the User and write it to the file
nextdata="go"
while len(nextdata) > 0:
    nextdata=input("Please enter a first and last name. To end, press
enter. ")
    fpointer1.write(nextdata)
    fpointer1.write("\n")

fpointer1.close()

fpointer2=open("DataFile.txt", "r")
fpointer3=open("DataFile.txt", "a")

locationf2=fpointer2.tell()
print("fpointer2 is pointing to location ",locationf2)

locationf3=fpointer3.tell()
print("fpointer3 is pointing to location ",locationf3)

for line in fpointer2:
```

```
    print("Line is: ",line)

locationf2=fpointer2.tell()
print("fpointer2 is pointing to location ",locationf2)

fpointer3.write("New Line 1\n")
fpointer3.write("New Line 2")

locationf3=fpointer3.tell()
print("fpointer3 is pointing to location ",locationf3)

# Close file before reading
fpointer3.close()

#Go to the start of the file first
fpointer2.seek(0)

#Read all the file lines into the variable called all_lines
all_lines=fpointer2.readlines()

#all_lines is a list. We can print the first item in the list using [0]
print(all_lines[0])

#Go back to the start of the file
fpointer2.seek(0)

#print all the lines of the file
counter=1
for line in fpointer2:
    print("This is file line ",counter, ": ",line)
    counter=counter+1

fpointer2.close()
```

Possible input and output for the above program example:

```
Please enter a first and last name. To end, press enter. John Smith

Please enter a first and last name. To end, press enter. Andy Donner

Please enter a first and last name. To end, press enter. Pat Gomez

Please enter a first and last name. To end, press enter.
fpointer2 is pointing to location  0
fpointer3 is pointing to location  38
Line is:  John Smith

Line is:  Andy Donner

Line is:  Pat Gomez

Line is:

fpointer2 is pointing to location  38
fpointer3 is pointing to location  60
John Smith
```

```
This is file line  1 :  John Smith

This is file line  2 :  Andy Donner

This is file line  3 :  Pat Gomez

This is file line  4 :

This is file line  5 :  New Line 1

This is file line  6 :  New Line 2
```

Review and discussion of the program by line number:

```
1.  # Data_I_O_example.py
2.  # Ami Gates
3.  # Create a new file for writing
4.  fpointer1=open("DataFile.txt", "w")

5.  # Get data from the User and write it to the file
6.  nextdata="go"
7.  while len(nextdata) > 0:
8.      nextdata=input("Please enter a first and last name. To end, press enter. ")
9.      fpointer1.write(nextdata)
10.     fpointer1.write("\n")

11. fpointer1.close()

12. fpointer2=open("DataFile.txt", "r")
13. fpointer3=open("DataFile.txt", "a")

14. locationf2=fpointer2.tell()
15. print("fpointer2 is pointing to location ",locationf2)

16. locationf3=fpointer3.tell()
17. print("fpointer3 is pointing to location ",locationf3)


18. for line in fpointer2:
19.     print("Line is: ",line)

20. locationf2=fpointer2.tell()
21. print("fpointer2 is pointing to location ",locationf2)

22. fpointer3.write("New Line 1\n")
23. fpointer3.write("New Line 2")

24. locationf3=fpointer3.tell()
25. print("fpointer3 is pointing to location ",locationf3)

26. # Close the write pointer before reading the file
27. fpointer3.close()

28. #Go to the start of the file first
29. fpointer2.seek(0)
```

```
30. #Read all the file lines into the variable called all_lines
31. all_lines=fpointer2.readlines()

32. #all_lines is a list. We can print the first item in the list using [0]
33. print(all_lines[0])

34. #Go back to the start of the file
35. fpointer2.seek(0)

36. #print all the lines of the file
37. counter=1
38. for line in fpointer2:
39.     print("This is file line ",counter, ": ",line)
40.     counter=counter+1

41. fpointer2.close()
```

It is also recommended that you type in and run the program.

Lines 1 – 3:
The first three lines of code are comments that offer the name, author, and goal of the program.

Line 4:

```
fpointer1=open("DataFile.txt", "w")
```

This line of code creates a file handle called `fpointer1`. Recall that a file handle is a variable (or pointer) that allows you to manage a file. Like all variable names in Python, the name of a file handle can be anything within the rules of naming.

Within this line of code, the `open()` function is used to open the file named "DataFile.txt". The file in this example program is being opened/created with the "w" mode. The "w" mode is generally only used to create a file, or to open and delete all contents of a file. As a word of warning, opening a file that already exists with the "w" mode will delete the file and open it as an empty file.

If the file already exists, and the goal is to read and write to the file, it is better to use "r+" for reading and writing, or "a" for append. As a side note, file names must be exact but are **not case sensitive**, which means that the capitalization does not matter when creating and referencing files.

Lines 5 – 11:

The next several statements in the program are the following:

```
# Get data from the User and write it to the file
nextdata="go"
while len(nextdata) > 0:
    nextdata=input("Please enter a first and last
name. To end, press enter. ")
```

```
            fpointer1.write(nextdata)
            fpointer1.write("\n")

        fpointer1.close()
```

These statements work together to allow the user to write new data into the file. The variable called `nextdata` is created and initialized to the value *"go".* Then, a *while loop* is used to allow user to enter as much data as they wish. Inside the while loop, the **condition** for the loop is the length of whatever the variable `nextdata` is equal to. The first time the while loop is entered, the variable `nextdata` is equal to "*go*", which has a length of "2" (because it is two characters long). Therefore, the while loop will proceed because the condition, `len(nextdata) > 0` is **True**.

Once inside the while loop, the variable `nextdata` is set equal to whatever the user enters. The `input()` function is used to get the user input and store it into `nextdata`. Notice that the user is informed that he/she can end or stop by simply pressing "enter".

Why will the loop end when the user presses only "enter"? If the user does not enter anything and simply presses "enter" (or return) on his/her computer, then the value in `nextdata` will have no length because it will contain nothing. Therefore, when the while loop checks to see if the length of `nextdata` is greater than 0, the result will be **False** and the while loop will end.

Each time the user enters data, it is collected into `nextdata` and then is written into the file using the write statement:

```
fpointer1.write(nextdata)
```

Next, the statement,

```
fpointer1.write("\n"),
```

will write a **new line (carriage return)** into the file. Writing a new line into the file is the same a pressing enter or return when typing in a word processing document. It explicitly creates a new line in the file.

Once the while loop ends and all the user input is written to the file, the file must be closed. Because the file was opened using "w", it cannot be read without first closing it and then opening it with either "r" (for read) or "r+" (for read or write).

To close the file, the following statement is employed.

```
fpointer1.close()
```

At this point, the file is closed and it contains all the data the user has entered. It is recommended that you locate and open this file using the File explorer in Spyder and confirm its contents.

Lines 12 and 13:

The next two statements in the program are the following:

```
fpointer2=open("DataFile.txt", "r")
fpointer3=open("DataFile.txt", "a")
```

The first of these two statements opens the file for reading using "r". The file handle name for **reading** the file is `fpointer2`. The second statement opens the same file at the same time for appending (using "a"). The name of the **appending** file handle is `fpointer3`.

The goal of this is to illustrate that more than one file handle can be enabled for the same file at the same time. The danger of this is keeping track of where each file handle (file pointer) is located in the file. The seek() and tell() functions are ideal for this.

For example, as the file is read from, the read file handle, called `fpointer2` will move accordingly. Similarly, as the file is written to, the file handle called `fpointer3` will move accordingly.

At this moment in the program, both `fpointer2` and `fpointer3` are at location 0 (the very beginning of the file).

Lines 14 – 17:

The next four statements show the location of these file handles. The `tell()` function is used to gain this information.

```
locationf2=fpointer2.tell()
print("fpointer2 is pointing to location ",locationf2)

locationf3=fpointer3.tell()
print("fpointer3 is pointing to location ",locationf3)
```

The `tell()` function will tell you where the file handle is pointing in the file; it returns an integer number that is the exact character position of the file handle at that moment given that "0" is the start of the file.

Notice that the variables, `locationf2` and `locationf3` will be assigned the locations of each file handle. By viewing the output for the program above, you can confirm that the location of `fpointer2` is 0 (the start of the file) and the location of `fpointer3` is "38". Why is it 38?

To answer this, first notice that the file has already been written to. Viewing the input/output example above (or your version of the file if you are creating each example), you can see that three names, John Smith, Andy Donner, and Pat Gomez, have been added to the file thus far in the program, as well as a new line (\n) character. Each new line is a "\n" and so counts as two character locations in the file. Therefore, when the file is opened for appending via,

`fpointer3=open("DataFile.txt", "a")`, the location of the file handle (the pointer) will be right after the very last item that had been added, which was a new line. If you count, starting from 0, you get 11 for John Smith (including the space between John and Smith and the new line after Smith), you get 13 for Andy Donner, you get 11 for Pat Gomez, and 2 for the extra newline that follows the three names. This adds to 37 and the very next location is 38.

Figure 8.3 will illustrate the contents of the file as well as an example run for the program.

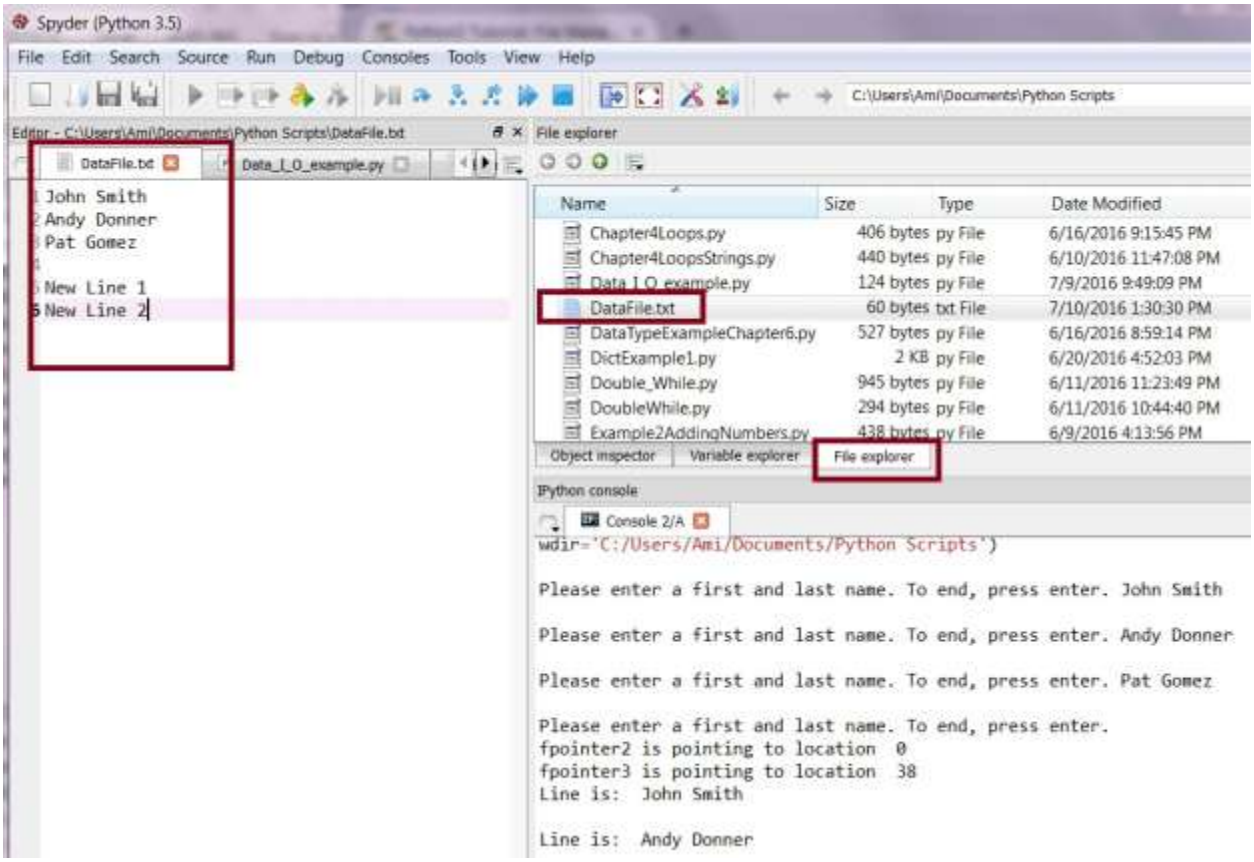

Figure 8.3: Example of the Program Data_I_O_example.py above.

Lines 18-21:

Continuing the discussion of program *Data_I_O_example.py* above, this program uses three different file handles named `fpointer1`, `fpointer2`, and `fpointer3`, each for different tasks. It is possible to use as many file handles for the same file as desired. However, before reading from the file, it is best to close the file handles for appending or writing. For this reason, notice in the program that `fpointer1.close()` occurs once the initial write is completed. Similarly, `fpointer3.close()` occurs before the reading starts.

Lines 18 - 21 are:

```
for line in fpointer2:
    print("Line is: ",line)
locationf2=fpointer2.tell()
print("fpointer2 is pointing to location ",locationf2)
```

This for/in loop allows the program to loop through each new line of the file. In this case, the variable, `line` holds all the data in the given line. The `print` statement prints each line of the file as it is read in. The for/in loop is one way to read the contents of a file, but it is not the only option. We will see an alternative shortly.

The tell function is used to record the current location of `fpointer2` into variable `locationf2`. This information is then printed as output.

Lines 22 – 27:

The next several lines of code are the following:

```
fpointer3.write("New Line 1\n")
fpointer3.write("New Line 2")

locationf3=fpointer3.tell()
print("fpointer3 is pointing to location ",locationf3)

# Close file before reading

fpointer3.close()
```

Here, the file handle called `fpointer3`, which was used to open the file for appending, is being used to write further information into the file without overwriting (erasing) what is already there. The append method is used to add information to a file. Information will be added at the location of the file handle (where the file handle is pointing in the file at that moment). The location of the file handle can be determined using `tell()` and changed using `seek(n)`.

Code lines 22 and 23 add two additional lines of information to the file. The first line that is added to the file is `"New Line 1\n"`, and the second line of information added is `"New Line 2"`. As a point of interest, the location of the file handle `fpointer3` is again collected. This is done and printed in code lines 24 and 25.

Notice in the output shown above, that the location of the file handle has changed because additional data has been written to the file. The new location is 60. Once the writing to the file is completed, the file is closed (code line 27).

Before moving forward, notice that two of the three file handles are now closed. The only pointer left open and active is `fpointer2`. Recall that `fpointer2` was opened for reading.

Lines 28 – 33:

```
#Go to the start of the file first
fpointer2.seek(0)
#Read all the file lines into the variable called
all_lines
all_lines=fpointer2.readlines()
#all_lines is a list. We can print the first item in
the list using [0]
print(all_lines[0])
```

Here, the `seek()` function is used to move the file handle (the pointer in the file) to a different location within the file. Code line 29, `fpointer2.seek(0)`, moves the file handle (called fpointer2) to location "0" in the file (which is the beginning). This is important and helpful when you wish to read the entire contents of the file, or return to the beginning of the file.

Code line 31 uses the `readlines()` function to read **all** lines in the file into the **list** variable called `all_lines`. The `readlines()` function is not the same as the `readline()` function.

The **readlines** function will read all of the lines of the entire file, starting wherever the file handle is pointing. The **readline** function will read one line only at a time and can be used within a for/in loop.

Because `all_lines` is a list structure, the statement code line 33, `print(all_lines[0])`, will print "John Smith" (which can be verified in the above example output) because `all_lines[0]` represents the first line of the file. Using this same logic, `all_lines[1]` would print "Andy Donner" in this case – the second line of the file.

Lines 34 – 40

The last several lines in the program are the following:

```
#Go back to the start of the file
fpointer2.seek(0)
#print all the lines of the file
counter=1
for line in fpointer2:
    print("This is file line ",counter, ": ",line)
    counter=counter+1

fpointer2.close()
```

Recall again that the only file handle currently open and in action is `fpointer2`. Recall also that whenever a file is read or written to, the file handle moves. Therefore, the current location of

`fpointer2` is not at the beginning of the file. The `tell()` function can always be used to determine the current location of the file handle (where it is pointing in the file). In addition, the `seek(0)` function can always be used to return to the beginning of the file.

Code line 35, `fpointer2.seek(0)`, will move the file handle, `fpointer2`, so that it points to the beginning of the file again (location 0). In code line 37, a variable called `counter` is created to keep track of how many lines in the file have been read. At first, `counter` is set to 1. Next, the for/in loop is used to read each line in the file (code lines 38 – 40). The loop will automatically end on its own when the end of the file is reached.

Each `line` in the file is printed (code line 39) and the `counter` variable is **incremented** to keep track of the number of lines read from the file. The last line of code (line 40) is the statement, `fpointer2.close()`. It is critical to close all files that are no longer in use.

Using the "with" method for File Management

An alternative method for managing files in Python 3 is via the "with" keyword.

```
#WithOpenExampleCh8.py
#Author Ami Gates

with open("MyNewFile.txt", "w") as f:
    f.write("Hello World\n")

with open("MyNewFile.txt", "r") as f:
    line=f.read()

#This will print True because the open with method
# closes the file as well
print(f.closed)
```

Using the `with` keyword has the benefit of assuring that the file opened will be properly and implicitly closed, even if an exception is thrown during processing. Once a file is open, whether using the `with` keyword or the more basic method, the remaining options are the same. This next example will illustrate.

```
#WithOpenExampleCh8.py
#Author Ami Gates

with open("MyNewFile.txt", "w") as f:
    f.write("Hello World\n")
    print(f.tell())
    f.write("This is another line to be written\n")
#This will print True
```

```
print(f.closed)

with open("MyNewFile.txt", "r") as f:
    #This will print all lines in the file
    print(f.readlines())
    #This returns the file pointer to 0 (the start of the
file)
    f.seek(0)
    #This prints the current location of the file pointer,
which is 0
    print(f.tell())
    #This prints each line of the file
    for line in f:
        print(line)

#This will print True because the open with method
# closes the file as well
print(f.closed)
```

Summary of File Methods

1) Create a new file.

```
file1=open("MyNewFile.txt", "w")
```

Notes: The *file1* is known as the file handle or variable that controls the file operations. The *MyNewFile.txt* is the actual name of the file itself. This is a .txt or text file. If you look in the File Explorer, you will see the new file name there. The *"w"* stands for **write only**.

2) Write data or information into the file.

```
file1.write("Hello World\n")
```

Notes: This will write the words, “Hello World”, into the file as well as a newline.

3) Close the file.

```
file1.close()
```

Notes: Always close a file when it is not in use.

4) Open and file for reading or writing and read from the file. The mode “r+” is for reading or writing. The “r” option is for read only.

```
file1=open("MyNewFile.txt", "r+")
```

5) Read each line from a file.

```
NextLine=file1.readline()
```

6) Read all lines from a file into a list

```
ListOfAllLines=file1.readlines()
```

7) Read the entire file (including the newlines) into a variable

```
WholeFileContents=file1.read()
```

8) To return the current character location of the file handle within the file (where 0 is the first character in the file).

```
file1.tell()
```

9) Move the file handle to a specific location in the file (by character number n).

```
file1.seek(n)
```

10) Append data to an existing file rather than overwrite the file. The mode option, “a”, opens the file for appending

```
file1=open("MyNewFile.txt", "a")
```

11) Using “with” allows a file to be implicitly closed, even in the case of an exception during processing.

```
with open("MyNewFile.txt", "r") as File:
     #scope of the with…
     data=File.read()
     …


with open("MyNewFile.txt", "w") as File:
     #scope of the with…
     data=File.write()
     …
```