In this session, we'll cover the process of working with files using the Python programming language. The built-in methods it offers makes it very easy to handle files using a relatively small amount of code. As with anything in programming, there are many ways to achieve the same goal when it comes to files, but in this document, we'll stick with the basics and show the most common ways to perform these actions.

## **Python Files**

A file simply is a collection of data stored in the form of a sequence of bytes in a machine.

It’s a collection of data or information stored in the memory heap that has a name called filename.

There are two types of files in Python.

1. **Text Files**
2. **Binary Files**

### ****Text files in Python****

[Text files](https://en.wikipedia.org/wiki/Text_file) are humanly readable which is displayed in a sequence of characters that can be easily interpreted by humans as it presented in the textual form. Common editors like notepad and others can be used to interpret and edit them.

They can be stored in plain text (.txt) format or rich text format (.rtf).

### ****Binary files in Python****

In [binary files](https://en.wikipedia.org/wiki/Binary_file) data is displayed in some encoded format (using 0’s and 1’s) instead of plain characters. Typically they contain the sequence of bytes.

They are stored in .bin format.

Any file operation can be broken down into three major steps:

1. Opening a file
2. Perform different file operations (Read/Write)
3. Closing a file

**Creating Files in Python**

**File Opening Modes**

There are modes in which you can open a file in Python. The mode you choose depends on how you plan to use the file, or what kind of data you'll be reading (writing) from (to) the file. This mode is specified when opening a file using the built-in open() method, explained in further detail in the next section.

Let's take a look at some of the possible combinations of file modes:

* w: Opens a file for writing and creates a new file if it doesn't yet exist. In the case that the file does exist, it overwrites it.
* w+: Opens a file for writing but also for reading and creating it if it doesn't exist. If a file already exists, it overwrites it.
* r: Opens a file for reading only.
* rb: Opens a file for reading in Binary format.
* wb: Opens a file for writing in Binary format.
* wb+: Opens a file for writing and reading in Binary format.
* a: Opens a file for appending at the end of the file.
* +: In general, this character is used along side r, w, or a and means both writing and reading.

If no file mode is specified, then r will be assumed by default.

When choosing a mode, take in to careful consideration what your use-case is and what all the file will be used for for the duration that it is open.

### Python Open File

According to the previous discussion, the first step we have to perform in Python File Operation is opening that file. You can open a file by using open() function. This function take two arguments. The first one is file address and the other one is opening mode. There are some mode to open a file. Most common of them are listed below:

#directory: /desktop/sibsankar/filecopyCode.py

text\_file = open(myFile.txt','r')

#Another method using full location

text\_file2 = open('/home/sibsankar/myFile.txt','r')

print('First Method')

print(text\_file)

print('Second Method')

print(text\_file2)

### Python Read File, Python Write File

There are some methods to read from and write to file. The following list are the common function for read and write in python. Note that, to perform read operation you need to open that file in read mode and for writing into that file, you need to open that in write mode. If you open a file in write mode, the previous data stored into that fill will be erased.

* **read**() : This function reads the entire file and returns a string
* **readline**() : This function reads lines from that file and returns as a string. It fetch the line n, if it is been called nth time.
* **readlines**() : This function returns a list where each element is single line of that file.
* **readlines**() : This function returns a list where each element is single line of that file.
* **write**() : This function writes a fixed sequence of characters to a file.
* **writelines**() : This function writes a list of string.
* **append**() : This function append string to the file instead of overwriting the file.

The following code will guide you to read from file using Python File Operation. We take ‘myfile.txt’ as our input file.

#open the file

text\_file = open('/home/sibsankar/ myfile.txt','r')

#get the list of line

line\_list = text\_file.readlines();

#for each line from the list, print the line

for line in line\_list:

print(line)

text\_file.close()

**Another way:**

#open the file

text\_file = open('/home/sibsankar/ myfile.txt','w')

#initialize an empty list

word\_list= []

#iterate 4 times

for i in range (1, 5):

print("Please enter data: ")

line = input() #take input

word\_list.append(line) #append to the list

text\_file.writelines(word\_list) #write 4 words to the file

text\_file.close() #don’t forget to close the file

### Moving Files in Python

To move a file in Python, we will need to import the os and shutil modules that provide us the ability to copy, move, and remove files in Python. Both of these modules provide methods to do so, although in many cases the shutil module has more convenient methods.

import os

import shutil

# Move a file by renaming it's path

os.rename('/home/sibsankar/d1/xfile.txt', '/home/sibsankar /d2/xfile.txt')

# Move a file from the directory d1 to d2

shutil.move('/home/sibsankar /d1/xfile.txt', '/home/sibsankar /d2/xfile.txt')

Keep in mind that the destination directory needs to exist in order for this example to work. Once you've set up the directories "d1" and "d2" (or just changed the example to fit your directory structure), run the code. Now check out the "d2" directory and you should see the xfile.txt if now present in that directory.

### Python Copy File

We can use shutil to copy file. Below is an example showing two different methods to copy file.

import shutil

shutil.copy2('/home/sibsankar/abc.txt', '/home/sibsankar/abc\_copy2.txt')

#another way to copy file

shutil.copyfile('/ home/sibsankar/abc.txt', '/ home/sibsankar/abc\_copyfile.txt')

print("File Copy Done")

### Deleting Files in Python

As you probably guessed, it's pretty easy to remove a file in Python using the **remove**() method from the os module.

In our example below, we'll delete the "sibs.txt". All we need to do is call the remove() method with the path of the file we want to delete:

import os

# Delete sibs.txt

os.remove('/home/sibsankar/d2/sibs.txt') Now, check out the "d2" directory again and the file sibs.txt will now be done. Simple as that!

### Python Close File

As you see in the previous example, we used **close()** function to close the file. Closing the file is important.

text\_file.close()

### Python FileNotFoundError

You will get this error if the file or directory is not present. A sample stack trace is given below.

File "/home/sibsankar/Desktop/abc.py", line 2, in <module>

text\_file = open('/home/sibsankar /Desktop/xyz.txt','r')

***FileNotFoundError: [Errno 2] No such file or directory: '/home/sibsankar /Desktop/xyz.txt'***

Please check the file path and correct it to get rid of FileNotFoundError.

### Conclusion

In this document, we showed very simple examples of how to create, move, and delete files in Python using the built-in functions such as open(), shutil.move(), and os.remove(). In addition, we presented a simple introduction and explanation of Python file modes.