

File Handling in Python

File handling is an important activity in every web app. The types of activities that you can perform on the opened file are controlled by Access Modes. These describe how the file will be used after it has been opened.

These modes also specify where the file handle should be located within the file. Similar to a pointer, a file handle indicates where data should be read or put into the file.

In Python, there are six methods or access modes, which are:

1. **Read Only ('r'):** This mode opens the text files for reading only. The start of the file is where the handle is located. It raises the I/O error if the file does not exist. This is the default mode for opening files as well.
2. **Read and Write ('r+'):** This method opens the file for both reading and writing. The start of the file is where the handle is located. If the file does not exist, an I/O error gets raised.

3. **Write Only ('w'):** This mode opens the file for writing only. The data in existing files are modified and overwritten. The start of the file is where the handle is located. If the file does not already exist in the folder, a new one gets created.
4. **Write and Read ('w+'):** This mode opens the file for both reading and writing. The text is overwritten and deleted from an existing file. The start of the file is where the handle is located.
5. **Append Only ('a'):** This mode allows the file to be opened for writing. If the file doesn't yet exist, a new one gets created. The handle is set at the end of the file. The newly written data will be added at the end, following the previously written data.
6. **Append and Read ('a+'):** Using this method, you can read and write in the file. If the file doesn't already exist, one gets created. The handle is set at the end of the file. The newly written text will be added at the end, following the previously written data.

Below is the code required to create, write to, and read text files using the Python file handling methods or access modes.

How to Create Files in Python

In Python, you use the `open()` function with one of the following options – "x" or "w" – to create a new file:

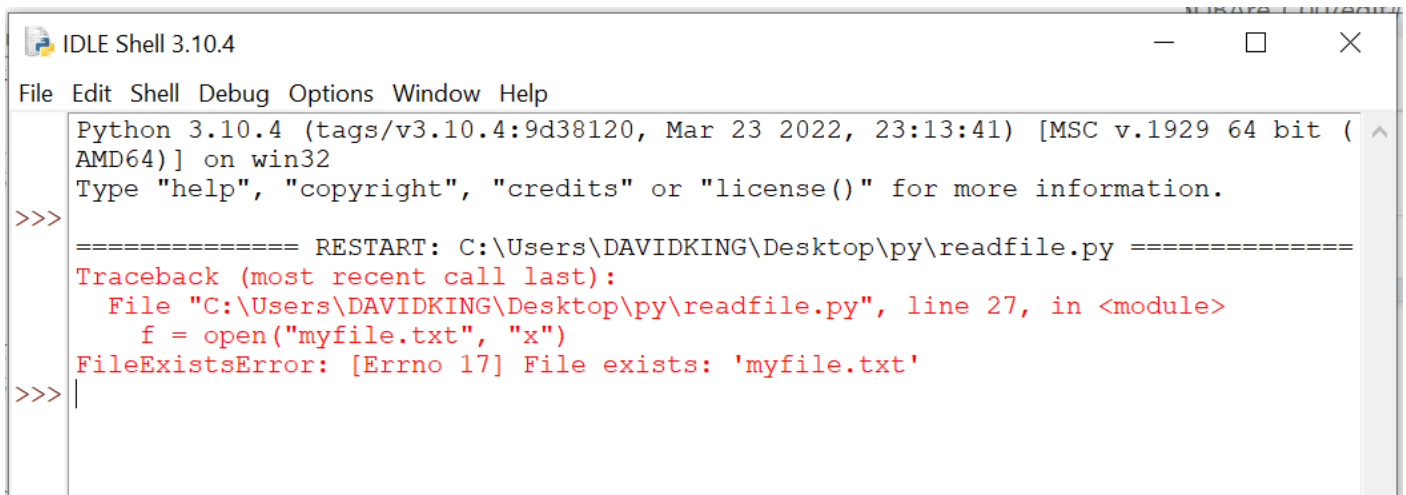
- **"x" – Create:** this command will create a new file if and only if there is no file already in existence with that name or else it will return an error.

Example of creating a file in Python using the "x" command:

```
#creating a text file with the command function "x"
```

```
f = open("myfile.txt", "x")
```

We've now created a new empty text file! But if you retry the code above – for example, if you try to create a new file with the same name as you used above (if you want to reuse the filename above) you will get an error notifying you that the file already exists. It'll look like the image below:



```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\DAVIDKING\Desktop\py\readfile.py =====
Traceback (most recent call last):
  File "C:\Users\DAVIDKING\Desktop\py\readfile.py", line 27, in <module>
    f = open("myfile.txt", "x")
FileExistsError: [Errno 17] File exists: 'myfile.txt'
>>> |
```

- **"w" – Write:** this command will create a new text file whether or not there is a file in the memory with the new specified name. It does not return an error if it finds an existing file with the same name – instead it will overwrite the existing file.

Example of how to create a file with the "w" command:

```
#creating a text file with the command function "w"
```

```
f = open("myfile.txt", "w")
```

```
#This "w" command can also be used create a new file but unlike
```

With the code above, whether the file exists or the file doesn't exist in the memory, you can still go ahead and use that code. Just keep in mind that it will overwrite the file if it finds an existing file with the same name.

How to Write to a File in Python

There are two methods of writing to a file in Python, which are:

The `write()` method:

This function inserts the string into the text file on a single line.

Based on the file we have created above, the below line of code will insert the string into the created text file, which is "myfile.txt."

```
file.write("Hello There\n")
```

The `writelines()` method:

This function inserts multiple strings at the same time. A list of string elements is created, and each string is then added to the text file.

Using the previously created file above, the below line of code will insert the string into the created text file, which is "myfile.txt."

```
f.writelines(["Hello World ", "You are welcome to Fcc\n"])
```

Example:

```
#This program shows how to write data in a text file.
```

```
file = open("myfile.txt","w")
```

```
L = ["This is Lagos \n","This is Python \n","This is Fcc \n"]
```

```
# i assigned ["This is Lagos \n","This is Python \n","This is Fcc \n"]
```

```
# Variable are containers in which values can be stored.
```

```
# The \n is placed to indicate the end of the line.
```

```
file.write("Hello There \n")
```

```
file.writelines(L)
```

```
file.close()
```

```
# Use the close() to change file access modes
```

How to Read From a Text File in Python

There are three methods of reading data from a text file in Python. They are:

The read() method:

This function returns the bytes read as a string. If no n is specified, it then reads the entire file.

Example:

```
f = open("myfiles.txt", "r")
```

```
#('r') opens the text files for reading only
```

```
print(f.read())
```

#The "f.read" prints out the data in the text file in the shell

The readline() method:

This function reads a line from a file and returns it as a string. It reads at most n bytes for the specified n. But even if n is greater than the length of the line, it does not read more than one line.

```
f = open("myfiles.txt", "r")  
print(f.readline())
```

The readlines() method:

This function reads all of the lines and returns them as string elements in a list, one for each line.

You can read the first two lines by calling `readline()` twice, reading the first two lines of the file:


```
f = open("myfiles.txt", "r")  
print(f.readline())  
print(f.readline())
```

How to Close a Text File in Python

It is good practice to always close the file when you are done with it.

Example of closing a text file:

This function closes the text file when you are done modifying it:

```
f = open("myfiles.txt", "r")  
print(f.readline())  
f.close()
```

The `close()` function at the end of the code tells Python that well, I am done with this section of either creating or reading – it is just like saying End.

Example:

The program below shows more examples of ways to read and write data in a text file. Each line of code has comments to help you understand what's going on:

```
# Program to show various ways to read and
# write data in a text file.

file = open("myfile.txt","w")

L = ["This is Lagos \n","This is Python \n","This is Fcc \n"]

#i assigned ["This is Lagos \n","This is Python \n","This is Fcc
#to variable L

#The \n is placed to indicate End of Line

file.write("Hello There \n")
file.writelines(L)
file.close()

# use the close() to change file access modes
```

```
file = open("myfile.txt","r+")  
print("Output of the Read function is ")  
print(file.read())  
print()
```

```
# The seek(n) takes the file handle to the nth  
# byte from the start.
```

```
file.seek(0)
```

```
print( "The output of the Readline function is ")  
print(file.readline())  
print()
```

```
file.seek(0)
```

```
# To show difference between read and readline
```

```
print("Output of Read(12) function is ")  
print(file.read(12))  
print()
```

```
file.seek(0)
```

```
print("Output of Readline(8) function is ")
```

```
print(file.readline(8))
```

```
file.seek(0)
```

```
# readlines function
```

```
print("Output of Readlines function is ")
```

```
print(file.readlines())
```

```
print()
```

```
file.close()
```

This is the output of the above code when run in the shell. I assigned "This is Lagos", "This is Python", and "This is Fcc" to "L" and then asked it to print using the "file.read" function.

The code above shows that the "readline()" function is returning the letter based on the number specified to it, while the "readlines()" function is returning every string assigned to "L" including the \n. That is, the "readlines()" function will print out all data in the file.



IDLE Shell showing the output of the program

Conclusion

Hopefully, after going through this tutorial, you should understand what file handling is in Python. We also learned the modes/methods required to create, write, read, and close() a text file using some basic examples from Python.