**Python File Handling**

Till now, we were taking the input from the console and writing it back to the console to interact with the user.

Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

The file handling plays an important role when the data needs to be stored permanently into the file. A file is a named location on disk to store related information. We can access the stored information (non-volatile) after the program termination.

The file-handling implementation is slightly lengthy or complicated in the other programming language, but it is easier and shorter in Python.

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

There are four different methods (modes) for opening a file:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exists

Syntax

To open a file for reading it is enough to specify the name of the file:

f = open("demofile.txt")

## **Open a File on the Server**

Assume we have the following file, located in the same folder as Python:

demofile.txt

Hello! Welcome to demofile.txt  
This file is for testing purposes.  
Good Luck!

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

### **Example**

f = open("demofile.txt", "r")  
print(f.read())

f the file is located in a different location, you will have to specify the file path, like this:

### **Example**

Open a file on a different location:

f = open("D:\\myfiles\welcome.txt", "r")  
print(f.read())

## **Read Only Parts of the File**

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

### **Example**

Return the 5 first characters of the file:

f = open("demofile.txt", "r")  
print(f.read(**5**))

## **Read Lines**

You can return one line by using the readline() method:

### **Example**

Read one line of the file:

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

By calling readline() two times, you can read the two first lines:

### **Example**

Read two lines of the file:

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

By looping through the lines of the file, you can read the whole file, line by line:

### **Example**

Loop through the file line by line:

f = open("demofile.txt", "r")  
for x in f:  
  print(x)

## **Close Files**

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

### **Example**

Close the file when you are finish with it:

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

## **Write to an Existing File**

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

### **Example**

Open the file "demofile2.txt" and append content to the file:

f = open("demofile2.txt", "a")  
f.write("Now the file has more content!")  
f.close()  
  
#open and read the file after the appending:  
f = open("demofile2.txt", "r")  
print(f.read())

# Python code to illustrate append() mode

file = open('geek.txt', 'a')

file.write("This will add this line")

file.close()

### **Example**

Open the file "demofile3.txt" and overwrite the content:

f = open("demofile3.txt", "w")  
f.write("Woops! I have deleted the content!")  
f.close()  
  
#open and read the file after the overwriting:  
f = open("demofile3.txt", "r")  
print(f.read())

## **Create a New File**

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

### **Example**

Create a file called "myfile.txt":

f = open("myfile.txt", "x")

Result: a new empty file is created!

### **Example**

Create a new file if it does not exist:

f = open("myfile.txt", "w")

# Python code to create a file

file = open('geek.txt','w')

file.write("This is the write command")

file.write("It allows us to write in a particular file")

file.close()

## **Delete a File**

To delete a file, you must import the OS module, and run its os.remove() function:

### **Example**

Remove the file "demofile.txt":

import os  
os.remove("demofile.txt")

### **Renaming the file**

The Python **os** module enables interaction with the operating system. The os module provides the functions that are involved in file processing operations like renaming, deleting, etc. It provides us the rename() method to rename the specified file to a new name. The syntax to use the **rename()** method is given below.

**Syntax:**

1. rename(current-name, new-name)

The first argument is the current file name and the second argument is the modified name. We can change the file name bypassing these two arguments.

**Example 1:**

import os  
  
*# rename file3.txt to file4.txt*os.rename("file3.txt", "file4.txt")

**Output:**

The above code renamed current **file3.txt** to **file4.txt**

## **The with statement**

The **with** statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files. It is used in the scenario where a pair of statements is to be executed with a block of code in between.

The syntax to open a file using with the statement is given below.

1. with open(<file name>, <access mode>) as <file-pointer>:
2. #statement suite

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the **with** statement in the case of files because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file, we don't need to write the **close()** function. It doesn't let the file to corrupt.

Consider the following example.

### **Example**

* wwith open("file.txt",'w') as f:  
   content = f.write("Yusuf");  
   f.close()  
    
  with open("file.txt",'a') as f:  
   f.write("Tushar")  
   f.close()  
    
  with open("file.txt",'r') as f:  
   print(f.read())

# Handling Python FileNotFoundError

If you have received the error “FileNotFoundError: [WinError 2] The system cannot find the file specified”, it means that there is no file present at the path you specified.

### **Solution – Python FileNotFoundError**

There are two ways in which you can handle FileNotFoundError.

* Use try-except and handle FileNotFoundError
* Check if file is present, and proceed with the file operation accordingly.

import os  
  
try:  
 os.remove('file3.txt')  
 print('The file is removed.')  
except FileNotFoundError:  
 print('The file is not present.')

import os  
if os.path.exists("demofile.txt"):  
  os.remove("demofile.txt")  
else:  
  print("The file does not exist")