PROGRAMMING WITH PYTHON

Karuna sheel

NIELIT kurukshetra

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, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

- However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling.

Opening a file

Python provides the open() function which accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like **reading, writing**, etc.

The syntax to use the open() function is given below.

file object = open(<file-name>, <access-mode>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

SN	ACCESS MODE	DESCRIPTION
1	r	It opens the file to read-only. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2	rb	It opens the file to read only in binary format. The file pointer exists at the beginning of the file.
3	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5	w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
6	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
7	w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.
8	wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.
9	a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.
10	ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.
11	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Let's look at the simple example to open a file named "file.txt" (stored in the same directory) in read mode and printing its content on the console.

Example
#opens the file demofile.txt in read mode

```
f = open("demofile.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)
```

The close() method

Once all the operations are done on the file, we must close it through our python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.

We can perform any operation on the file externally in the file system is the file is opened in python, hence it is good practice to close the file once all the operations are done.

The syntax to use the close() method is given below.

fileobject.close()

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()
```

Python File Write

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())
```

```
f=open("hello.txt",'a')
str=input('ENTER TEXT:')    #writing content from
cosole
str1=str+"\n"           #Adding line space
f.write(str1)
f=open("hello.txt",'r')
print(f.read())
f.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 appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Note: the "w" method will overwrite the entire file.

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!

Python Delete 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")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example

Check if file exists, then delete it:

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

Delete Folder

To delete an entire folder, use the os.rmdir() method:

Example
Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

Note: You can only remove empty folders.

# Python code to illustrate split() function

```
with open("file.txt", "r") as file:
    data = file.readlines()
    for line in data:
        word = line.split()
        print(word)
```

Python File Methods

Method	Description
close()	Closes an opened file. It has no effect if the file is already closed.
detach()	Separates the underlying binary buffer from the TextIOBase and returns it.
fileno()	Returns an integer number (file descriptor) of the file.
flush()	Flushes the write buffer of the file stream.
isatty()	Returns True if the file stream is interactive.
read(n)	Reads at most n characters from the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Reads and returns one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Reads and returns a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from= SEEK_SET)	Changes the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resizes the file stream to size bytes. If size is not specified, resizes to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Writes the string s to the file and returns the number of characters written.
writelines(lines)	Writes a list of lines to the file.