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 not volatile and can be accessed every time. Here, comes the need of file handling
* File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

In Python, a file operation takes place in the following order.

* Open a file
* Read or write (perform operation)
* Close the file

## How to open a file?

* Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

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

File\_object = open(<file-name>, <access-mode>)

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.

R+

* It opens the file to read and write both. The file pointer exists at the beginning of the file.

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.

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.

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.

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.

*#opens the file file.txt in read mode*fileptr = open(**"file.txt"**,**"r"**)  
**if** fileptr:   
 print(**"file is opened successfully"**)

…………………………………………………………………………………..

## **The close() method**

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

fileobject.close()

*# opens the file file.txt in read mode*fileptr = open(**"file.txt"**,**"r"**)  
**if** fileptr:   
 print(**"file is opened successfully"**)   
*#closes the opened file*fileptr.close()

…………………………………………………………………………………….

## **Reading the file**

* To read a file using the python script, the python provides us the read() method. The read() method reads a string from the file. It can read the data in the text as well as binary format.
* The syntax of the read() method is given below.

fileobj.read(<count>)

* Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.
* *#open the file.txt in read mode. causes error if no such file exists.*fileptr = open(**r"file.txt"**,**"r"**);  
  *#stores all the data of the file into the variable content*content = fileptr.read();  
  *# prints the type of the data stored in the file*print(type(content))  
  *#prints the content of the file*print(content)  
  *#closes the opened file*fileptr.close()

………………………………………………………………………………….

## **Read Lines of the file**

* Python facilitates us to read the file line by line by using a function readline(). The readline() method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.
* *#open the file.txt in read mode. causes error if no such file exists.*fileptr = open(**"file.txt"**,**"r"**);  
  *#stores all the data of the file into the variable content*content = fileptr.readline();  
  *# prints the type of the data stored in the file*print(type(content))  
  *#prints the content of the file*print(content)  
  *#closes the opened file*fileptr.close()

…………………………………………………………

## **Looping through the file**

*#open the file.txt in read mode. causes an error if no such file exists.*fileptr = open(**"file.txt"**,**"r"**);  
*#running a for loop***for** i **in** fileptr:   
 print(i) *# i contains each line of the file*

…………………………………………

## **Writing the file**

* To write some text to a file, we need to open the file using the open method with one of the following access modes.
* **a:** It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.
* **w:** It will overwrite the file if any file exists. The file pointer is at the beginning of the file.
* *#open the file.txt in append mode. Creates a new file if no such file exists.*fileptr = open(**"file.txt"**,**"a"**);  
  *#appending the content to the file*fileptr.write(**"\n"**)  
  fileptr.write(**"Python is the modern day language. It makes things so simple."**)  
  *#closing the opened file*fileptr.close();

……………………………………………………………………………..

*#open the file.txt in write mode.*fileptr = open(**"file.txt"**,**"w"**);  
*#overwriting the content of the file*fileptr.write(**"Python is the modern day language. It makes things so simple."**)   
*#closing the opened file*fileptr.close();

Now, we can check that all the previously written content of the file is overwritten with the new text we have passed.

………………………………………………………………………………….

## **Using with statement with files**

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

with open(<file name>, <access mode>) as <file-pointer>:

    #statement suite

**with** open(**"file.txt"**,**'r'**) **as** f:   
 content = f.read();   
 print(content)

………………………….

## **File Pointer positions**

Python provides the tell() method which is used to print the byte number at which the file pointer exists. Consider the following example.

*# open the file file2.txt in read mode*fileptr = open(**"file.txt"**,**"r"**)  
*#initially the filepointer is at 0*print(**"The filepointer is at byte :"**,fileptr.tell())  
*#reading the content of the file*content = fileptr.read();  
*#after the read operation file pointer modifies. tell() returns the location of the fileptr.*print(**"After reading, the filepointer is at:"**,fileptr.tell())

………………………………………………………………………

## **Modifying file pointer position**

In the real world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.

For this purpose, the python provides us the seek() method which enables us to modify the file pointer position externally.

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

file-ptr.seek(offset[, **from**)

The seek() method accepts two parameters:

**offset:** It refers to the new position of the file pointer within the file.

**from:** It indicates the reference position from where the bytes are to be moved. If it is set to 0, the beginning of the file is used as the reference position. If it is set to 1, the current position of the file pointer is used as the reference position. If it is set to 2, the end of the file pointer is used as the reference position.

*# open the file file2.txt in read mode*fileptr = open(**"file.txt"**,**"r"**)  
*#initially the filepointer is at 0*print(**"The filepointer is at byte :"**,fileptr.tell())  
*#changing the file pointer location to 10.*fileptr.seek(20);  
*#tell() returns the location of the fileptr.*print(**"After reading, the filepointer is at:"**,fileptr.tell())

……………………………………….

## **Python os module**

## **Renaming the file**

**import** os;

#rename file2.txt to file3.txt

os.rename("file2.txt","file3.txt")

……………………………………………………………..

## **Removing the file**

**import** os;

#deleting the file named file3.txt

os.remove("file3.txt")

…………………………………………………………….

## **Creating the new directory**

**import** os;

#creating a new directory with the name new

os.mkdir("new")

……………………………………………………………..

## **Changing the current working directory**

**import** os;

#changing the current working directory to new

os.chdir("new")

………………………………………………………………..

## **The getcwd() method**

**import** os;

#printing the current working directory

**print**(os.getcwd())

## **Deleting directory**

**import** os;

#removing the new directory

os.rmdir("new")

## **Writing python output to the files**

In python, there are the requirements to write the output of a python script to a file.

The **check\_call()** method of module **subprocess** is used to execute a python script and write the output of that script to a file.

File.py

temperatures=[10,-20,-289,100]

**def** c\_to\_f(c):

**if** c< -273.15:

**return** "That temperature doesn't make sense!"

**else**:

        f=c\*9/5+32

**return** f

**for** t **in** temperatures:

**print**(c\_to\_f(t))

…………………………………………….

**import** subprocess

with open("output.txt", "wb") as f:

    subprocess.check\_call(["python", "file.py"], stdout=f)

……………………………………………………………………………………………………

## **The file related methods**

The file object provides the following methods to manipulate the files on various operating systems.

|  |  |  |
| --- | --- | --- |
| **SN** | **Method** | **Description** |
| 1 | file.close() | It closes the opened file. The file once closed, it can't be read or write any more. |
| 2 | File.fush() | It flushes the internal buffer. |
| 3 | File.fileno() | It returns the file descriptor used by the underlying implementation to request I/O from the OS. |
| 4 | File.isatty() | It returns true if the file is connected to a TTY device, otherwise returns false. |
| 5 | File.next() | It returns the next line from the file. |
| 6 | File.read([size]) | It reads the file for the specified size. |
| 7 | File.readline([size]) | It reads one line from the file and places the file pointer to the beginning of the new line. |
| 8 | File.readlines([sizehint]) | It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function. |
| 9 | File.seek(offset[,from) | It modifies the position of the file pointer to a specified offset with the specified reference. |
| 10 | File.tell() | It returns the current position of the file pointer within the file. |
| 11 | File.truncate([size]) | It truncates the file to the optional specified size. |
| 12 | File.write(str) | It writes the specified string to a file |
| 13 | File.writelines(seq) | It writes a sequence of the strings to a file. |

………………………………………………………

## **Check if File exist:**

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")

…………………………………………………………………………

try:

f = open("test.txt",encoding = 'utf-8')

# perform file operations

finally:

f.close()

………………………………………………………………………..

Exercise

* Read the entire content of the file
* Write a Python program to read first n lines of a file.

fp=open(**"file.txt"**,**"r+"**)  
**for** line **in** range(3):  
 print(fp.readline(),end=**""**)  
fp.close()

………………………………………….

fp=open(**"file.txt"**,**"r+"**)  
c=0  
**for** i **in** fp:  
 **if** c<3:  
 print(i,end=**""**)  
 c+=1  
fp.close()

……………………………………………

READ FILE LINES AND STORE AS LIST

**with** open(**"file.txt"**) **as** f:  
*#Content\_list is the list that contains the read lines.* content\_list = f.readlines()  
 print(content\_list)

……………………………………………………………..

COUNT THE NUMBER OF LINES