File Management / input output in python

Created By :-

The EasyLearn Academy

9662512857

What is file?

- 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, pen drive/ compact disc (CD)).
- Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.
- When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.
- Hence, in Python, a file operation takes place in the following order.
- Open a file
- Read or write (perform operation)
- Close the file

MORE on file management

- File management concept in python is easy and short.
- Python treats file differently as text or binary and this is important.
- Each line of code includes a sequence of characters and they form text file.
- Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character.
- It ends the current line and tells the interpreter a new one has begun.
- Python has several functions for creating, reading, updating, and deleting files.

How to create a new file/ open existing file?

- Open() function is used to create a new file and it can also be used to open existing file.
- This function creates a **file** object, which would be utilized to call other support methods associated with it.
- **Syntax**
- file object = open(file_name [, access_mode][, buffering])
- Here are parameter details −
- **file_name** − The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode** − The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. This is optional parameter and the default file access mode is read (r).
- **buffering** − If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Modes of opening file.

r

Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

rb

Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

r+

Opens a file for both reading and writing. The file pointer placed at the beginning of the file.

rb+

Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

w

Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

wb

Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

w+

Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

wb+

Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

a

Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

ab

Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

a+

Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

ab+

Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Example of reading data from file

```
# a file named "list.txt", will be opened
  with the reading mode.
file = open('list.txt', 'r')
# This will print every line one by one in
  the file
for line in file:
  print (line)
```

Read() method

- There is more than one way to read a file in Python.
- If you need to extract a string that contains all characters in the file then you can use **file.read()**.

```
file = open("list.txt", "r")
print (file.read())
file.close()
```

- if we do not provide any argument into the file it reads whole file (all content) and print it on screen.
- We can also provide optional argument in read method that should be number which specifies no of character to be read from file.

Write into a File

- Writing to a file is as easy as reading from a file.
- To open a file for writing we set the second parameter to "w" instead of "r".
- To actually write the data into this file, we use the method write() of the file handle object.
- Let's start with a very simple and straightforward example:

```
file = open("example.txt", "w")
file.write("this is some text to be written into
  file \n")
file.close()
```

- Especially if you are writing to a file, you should never forget to close the file handle again.
- Otherwise you will risk to end up in a non consistent state of your data.

Example for simultaneously reading and writing:

```
FileReader = open(“source.txt","r")
FileWriter = open(“destination.txt","w")
for line in FileReader:
    print(line,end=‘’)
    FileWriter.write(line)
FileReader.close()
FileWriter.close()
```

Renaming and Deleting Files

- Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files.To use this module, you need to import it first and then you can call any related functions.
- **The rename() Method**
- The rename() method takes two arguments, the current filename and the new filename.

  Syntax

  os.rename(current_file_name, new_file_name)

  Example

  Following is an example to rename an existing file test1.txt –

```
import os

# Rename a file from test1.txt to test2.txt
os.rename( "test1.txt", "test2.txt" )
```

- **The remove() Method**
- You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.

  Syntax

  os.remove(file_name)

  Example

  Following is an example to delete an existing file test2.txt –

```
import os

# Delete file test2.txt
os.remove("text2.txt")
```

Directories in Python

- All files are stored within various directories, so you need to learn how work with directories like you learnt how to work with files.. The os module has several methods that help you create, remove, and change directories.
- mkdir()
- You can use the mkdir() to create directory in the current directory. You need to supply an argument to this method, which contains the name of the directory to be created.
- Syntax
- os.mkdir("python")
- chdir()
- You can use the chdir() method to change the current working directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.
- Syntax
- os.chdir("python")
- getcwd()
- The getcwd() method displays the current working directory.
- Syntax
- os.getcwd()

rmdir() Method

- The *rmdir()* method deletes the exiting directory, which is passed as an argument in the method.
- Before removing a directory, all the contents in it should be removed.
- **Syntax**
- os.rmdir(‘python')