CLASS-XII

COMPUTER SCIENCE

FILE HANDLING

(TEXT, BINARY AND CSV FILES)

NOTES

A file is a sequence of bytes on the disk/permanent storage where a group of related data is stored. File is created for permanent storage of data.

In programming, Sometimes, it is not enough to only display the data on the console. Those data are to be retrieved later on, then the concept of file handling comes. 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 file system which is not volatile and can be accessed every time. Here, comes the need of file handling in Python.

File handling in Python enables us to create, update, read, and delete the files stored on the file system through our python program. The following operations can be performed on a file.

In Python, File Handling consists of following three steps:

- Open the file.
- Process file i.e. perform read or write operation.
- Close the file.

Types of File

There are two types of files:

Text Files- A file whose contents can be viewed using a text editor is called a text file. A text file is simply a sequence of ASCII or Unicode characters. Python programs, contents written in text editors are some of the example of text files.

Binary Files-A binary file stores the data in the same way as as stored in the memory. The .exe files, mp3 file, image files, word documents are some of the

examples of binary files. We can't read a binary file using a text editor.

DIFFERENCE BETWEEN TEXT FILE AND BINARY FILE

Text File	Binary File
Its Bits represent character.	It represents a custom data.
Less prone to get corrupt as change soon as made and can be undone.	Can easily get corrupted, corrupt on reflects as even single bit change
Store only plain text in a file.	Can store different types of data
Widely used file format and can be opened in any text editor.	Developed for an application and can be can be opened in that only.
opened in any text editor. Mostly .txt and .rtf are used as extensions to text files.	Can have any application defined extensions.

Opening and Closing Files : To perform file operation, it must be opened first then after reading, writing, editing operation can be performed. To create any new file then too it must be opened. On opening of any file, a file relevant structure is created in memory as well as memory space is created to store contents. Once we are done working with the file, we should close the file.

Closing a file releases valuable system resource. In case we forgot to close the file, Python automatically close the file when program ends or file object is no longer referenced in the program. However, if our program is large and we are reading or writing multiple files that can take significant amount of resource on the system. If we keep opening new files carelessly, we could run out of resources. So be a good programmer, close the file as soon as all task are done with it.

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

File opening modes

Sr. No	Mode & Description
1	r - reading only. Sets file pointer at beginning of the file. This is the default mode.
2	rb – same as r mode but with binary file.
3	r+ - both reading and writing. The file pointer placed at the beginning of the file.
4	rb+ - same as r+ mode but with binary file.
5	w - writing only. Overwrites the file if the file exists. If not, creates a new file for writing.
6	wb – same as w mode but with binary file.
7	w+ - both writing and reading. Overwrites. If no file exists, creates a new file for R & W.
8	wb+ - same as w+ mode but with binary file.
9	a -for appending. Move file pointer at end of the file.Creates new file for writing,if not exist.
10	ab – same as a but with binary file

open() Function:

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`

"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

In addition you can specify if the file should be handled as binary or text mode

"t" - Text - Default value. Text mode

"b" - Binary - Binary mode (e.g. images)

Syntax

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

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

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Open a File on the Server

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

```
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())
```

If 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))
```

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())
```

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())
```

Python Delete File

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")
```

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")
```

```
# Program to Create Binary File and storing Few Names
size_of_rec = 20  #Each name will occupy 20 bytes
with open('Names.dat','wb') as f:
        ans='y'
        while ans.lower()=='y':
                name = input("Enter Name :")
                l = len(name)
                name = name + (size_of_rec-l)*' '
                # To add extra space with name to make it of length 20
                name = name.encode()
                f.write(name)
                ans=input("Add More ?")
```

```
Enter Name :AMIT
Add More ?y
Enter Name :VIKAS
Add More ?y
Enter Name :SUMIT
Add More ?y
Enter Name :NITIN
Add More ?y
Enter Name :SANJAY
Add More ?n
```

```
# Program to Update Name in Binary File
import os
size_of_rec = 20
# Finding Size of File
size = os.path.getsize('Names.dat')
print("Size of file : ",size)

#Finding Number of Records
num_rec = int(size / size_of_rec)
print("Number of Records :",num_rec)

with open('Names.dat','r+b') as f:
        old_name = input('Enter Name :')
        old_name = old_name.encode()

        new_name = input('Enter New Name :')
        ln = len(new_name)
        new_name = new_name + (20-ln)* ' '
        new_name=new_name.encode()

        position = 0
        found = False
        for i in range(num_rec):
                f.seek(position)
                str = f.read(20)  #Read each name
                if old_name in str:
                        print('Updated Record No. ',(i+1))
                        found=True
                        f.seek(-20,1)  #sending cursor 20 bytes back for update
                        f.write(new_name)
                position+=size_of_rec
        if not found:
                print('Name Not Found')
```

```
Size of file :  100
Number of Records : 5
Enter Name :SUMIT
Enter New Name :SHIKHAR
Updated Record No.  3
```

```
Size of file :  100
Number of Records : 5
Enter Name :JAYANTILAL
Enter New Name :JETHALAL
Name Not Found
```

```
#Program to search for any name in file and display the record
#number that contains the name
import os
size_of_rec = 20
# Finding Size of File
size = os.path.getsize('Names.dat')
print("Size of file : ",size)

#Finding Number of Records
num_rec = int(size / size_of_rec)
print("Number of Records :",num_rec)

with open('Names.dat','rb') as f:
      n = input('Enter Name to Search ')
      n = n.encode()
      position = 0
      found = False
      for i in range(num_rec):
                  f.seek(position)
                  str = f.read(20)
                  if n in str:
                              print('Found at Record # ',(i+1))
                              found=True
                  position+=size_of_rec
      if not found:
                  print('Name Not Found')
```

CSV FILE (Comma separated value)

CSV (Comma Separated Values) is a file format for data storage which looks like a text file. The information is organized with one record on each line and each field is separated by comma.

CSV File Characteristics :

• One line for each record

• Comma separated fields

• Space-characters adjacent to commas are ignored

• Fields with in-built commas are separated by double quote characters.

When Use CSV?

• When data has a strict tabular structure

• To transfer large database between programs

• To import and export data to office applications, Qedoc modules

• To store, manage and modify shopping cart catalogue

CSV Advantages

• CSV is faster to handle

• CSV is smaller in size

• CSV is easy to generate

• CSV is human readable and easy to edit manually

• CSV is simple to implement and parse

• CSV is processed by almost all existing applications

CSV Disadvantages

• No standard way to represent binary data

• There is no distinction between text and numeric values

• Poor support of special characters and control characters

• CSV allows to move most basic data only. Complex configurations cannot be imported and exported this way

• Problems with importing CSV into SQL (no distinction between NULL and quotes)

Write / Read CSV FILE

Writing and reading operation from text file is very easy. First of all, we have to import csv module for file operation/method call. For writing, we open file in 'w'

writing mode using open () method which create newFile like object.

Through csv.writer() method ,we create writer object to call writerow() method to write objects.

Similarly for reading, we open the file in ‘r’ mode and create newFile like object,further we create newfilereader object using csv.reader() method to read each row of the file.

Three file opening modes are there ‘w’,’r’,’a’. ‘a’ for append After file operation close, opened file using close () method.

EXAMPLE TO READ/WRITE DATA FROM CSV FILE

```
import csv
#csv file writing code
with open('d:\\a.csv','w') as newFile:
newFileWriter = csv.writer(newFile)
newFileWriter.writerow(['user_id','beneficiary'])
newFileWriter.writerow([1,'xyz'])
newFileWriter.writerow([2,'pqr'])
newFile.close()
#csv file reading code
with open('d:\\a.csv','r') as newFile:
newFileReader = csv.reader(newFile)
for row in newFileReader:
    print (row)
newFile.close()CSV FILE (Comma separated value)
```