Unit-4: Files

A file is a place on the disk where a group of related data is stored. Files are used for permanent storage of data. There are different kinds of files such as text files, image files, audio files, video files. Python gives easy ways to manipulate these files.

Generally, we divide files into two categories: Text files and Binary files
A Text file stores data in the form of alphabets, digits and other special symbols by storing their ASCII values and are in a human-readable format. for example, any file with a .txt, .py, etc extension. whereas Binary file contains a sequence or collection of bytes, and is readable only by computer.

In the text file, each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default. In the binary file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language.

File Input/Output Operations:

1) Open a file
2) Close a file
3) Read a file
4) Write a file

Creating and Reading of Text Data:

To read and write to a text file, we must open it.

Opening a file:

Syntax:

```
filepointer = open(filename, mode)
```

Filename can be text file with extension(.txt)

Modes:

"r" - Read - Opens a file for reading, error if the file does not exist (default value)

"a" - Append - Opens a file for appending

"w" - Write - Opens a file for writing/overwriting

“x" - Create - Creates the specified file, returns an error if the file exists ("x" is similar to "w". But for "x", if the file exists, raise **FileExistsError**)

Ex:

```
f = open(“demo.txt”, ’r’)
f = open(“demo.txt”, ’w’)
f = open(“demo.txt”, ’a’)
```

Closing a file:

Syntax:

```
filepointer.close()
```

Ex:

```
f = open(‘demo.txt’, ’r’)
print( f.read() )
f.close()
```

Reading a text file:
read() method is used for reading the content of the file.

```
#Write a python program to read data of demo.txt file
        f=open('demo.txt', 'r')
        print(f.read())
        f.close()

#Read only part of a file:
        f=open('demo.txt', 'r')
        print( f.read(3) )   #returns first three characters
        f.close()

#Read line:
        f=open('demo.txt', 'r')
        print( f.readline() )   #read only first line
        f.close()

#Read all lines at a time:
        f=open('demo.txt', 'r')
        print( f.readlines() )              #returns list of lines in the file
        f.close()
```

Writing into a file:
The ***write()*** method is used to write data to an empty file or existing file.
It has two modes:

"w" - Write - Opens a file for writing data into empty file (or) overwriting the data in the existing file

"a" - Append - Opens a file for appending the data at end of the file

Syntax:

```
        filepointer.write('Write the data which u need to append or overwrite')
```

Ex:

```
#Write a program to write data into sample.txt file
        f=open('sample.txt', 'w')
        f.write("welcome to python programming")
        f.close()

#To display the content of sample.txt file
        f=open('sample.txt', 'r')
        print( f.read() )
        f.close()
```

Output: welcome to python programming

```
#Write a program to write data into existing file sample.txt
        f=open("sample.txt", 'w')
        f.write("I am overwriting the existing content")
        f.close()
```

#To view the content in the sample.txt

```
f=open("sample.txt", 'r')
print( f.read() )
```

Output: I am overwriting the existing content

Write a program to append data to ***sample.txt*** file content

```
f=open("sample.txt", 'a')
f.write("python is very easy")
f.close()
```

#To view the content in the ***sample.txt***

```
f=open("sample.txt", 'r')
print( f.read() )
f.close()
```

Output: I am overwriting the existing content python is very easy

File handling methods():

1) **tell():** Returns current file pointer (***File Handle***) location in the file.
   This method takes no parameters and returns an integer value (***offset*** of the pointer from the beginning of the file). Initially file pointer points to the beginning of the file (if not opened in append mode). So, the initial value of ***tell()*** is zero.

Syntax: filepointer.tell()

Eg:

```
f=open('sample.txt','r')
print( f.tell() )          → 0
f.read(5)
print( f.tell() )          → 5
f.close()
```

2) **seek():** This method sets the filepointer position at the required offset.

Syntax: filepointer.seek(offset, from)

Where, offset – number of positions to move forward

from – it defines the point of reference (optional parameter)

(0- from the beginning of file, 1- from the current location, 2- from the end of file)

```
Eg:  seek(10)      #filepointer moves by position 10
     seek(10, 0)   #moving filepointer by 10 positions from begin of file
     seek(10,1)    #moving filepointer by 10 positions from current position
     seek(-10,2)   #moving filepointer backward by 10 positions from end of file
```

Note: non-zero cur-relative end-relative seeks are supported on Binary Files

3) **read()** – method reads the specified number of bytes from the file
4) **readline()** – will return a line from the file
5) **readlines()** – will return all the lines in the file in the form of a list

Reading and writing of binary files

Opening a file:

filepointer=open(filename, mode)

Filename with extension(.bin)

Modes:

"rb" - Read - Opens a binary file for reading

"wb" - write - Opens a binary file for writing /overwriting

Eg: f=open('demo.bin','rb')

```
# Creating demo.bin file
gedit demo.bin
10
20
a
b
c
#Write a python program to read the data in demo.bin file
        f=open('demo.bin','rb')
        print(f.read())
        f.close()

# creating empty.bin file
gedit empty.bin
# write a python program to write the data into empty.bin
        f=open('empty.bin','wb')
        l=[5,6,7]
        l1=bytearray(l)
        f.write(l1)
        f.close()

#to view the contents in empty.bin
        f=open('empty.bin','rb')
        print(f.read())
        f.close()

#To overwrite the content in existing file demo.bin
        f=open('demo.bin','wb')
        l=[5,6,7]
        l1=bytearray(l)
        f.write(l1)
        f.close()

#to view the contents in demo.bin
        f=open('demo.bin','rb')
        print(f.read())
        f.close()
```

CSV (Comma Separated Value) files

A CSV (comma-separated values) is a file containing text that is separated with a comma, It is a type of plain text that uses specific structuring to arrange tabular data. CSV files are used to handle large amount of data. Eg: Spreadsheets

Syntax:

col1,	col2,	col3
first row data1,	first row data2,	first row data3
second row data1,	second row data2,	second row data3

Reading and writing of CSV files

```
#create demo.csv file
gedit  demo.csv
SN,ROLLNO,NAME
1,571,kumar
2,568,ravi
```

#Write a python program to read the data in demo.csv

```
import csv
with  open('demo.csv', 'r') as file:
          reader = csv.reader(file)
          for row in reader:
                    print(row)
file.close()
```

Output:

```
['SN','ROLLNO','NAME']
['1','571','kumar']
['2','568','ravi']
```

#Creating an empty.csv file

```
gedit  empty.csv
```

#Writing the data into empty.csv file

```
import  csv
with  open("empty.csv","w")  as  file:
       f = csv.writer(file)
       f.writerow(['s.No', 'rollno', 'name'])
       f.writerow([1, 101, 'aaa'])
      f.writerow([2, 102, 'bbb'])
file.close()
```

#Writing the data into demo.csv file

```
import  csv
with  open("demo.csv", "w")  as  file:
      f = csv.writer(file)
      f.writerow(['s.No', 'rollno', 'name'])
      f.writerow([1,101, 'aaa'])
      f.writerow([2,102, 'bbb'])
file.close()
```

Additional Inputs on File Concepts:

File opening modes:

r – read	r+	-	read and write
w – write	w+	-	write and read
a – append	a+	-	append and read

#Example

```
f = open("sample.txt","w+")
f.write("aaa bbb ccc")
f.seek(0,0)
print(f.read())   → aaa bbb ccc
f.close()
```

pickle module in Python

Python ***pickle*** module is used for serializing and de-serializing a Python object structure. Python serialization is the act of converting a Python object into a byte stream.

Pickling" is the process whereby a Python object hierarchy is converted into a byte stream, and "unpickling" is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling is alternatively known as "serialization", or "flattening".

Note: Python has a more primitive serialization module called ***marshal***, but in general ***pickle*** should always be the preferred way to serialize Python objects.

#Example - pickle a list

```
import pickle
mylist = ['a', 'b', 'c', 'd']
with  open('datafile.pkl', 'wb') as f:
        pickle.dump(mylist, f)          #serializing into file
        var = pickle.dumps(mylist)   #serializing into a variable
f.close()
print(type(var))        →  <class 'bytes'>
print(var)              → serialized object gets printed
```

#Example - unpickle an object

```
import pickle
with  open('datafile.pkl', 'rb') as f:
        mylist = pickle.load(f)         #deserializing the object
f.close()
print(type(mylist))     → <class 'list'>
print(mylist)           → ['a', 'b', 'c', 'd']
```

os and *os.path* Modules

The ***os*** module in Python provides functions for interacting with the operating system.

The ***os.path*** module contains some useful functions on pathnames. These functions are used for different purposes such as for merging, normalizing and retrieving path names in Python.

Few important functions from ***os*** module:

os.getcwd() - returns the current working directory

os.mkdir(dirname) - creates a new directory

os.chdir(dirname) - it is used to change to the required directory

os.rmdir() - removes the specified directory

os.listdir() - returns the list of all files and directories in the specified directory

Following are few important functions of ***os.path*** module.

os.path.basename(path) - It is used to return the basename of the file (file name from the path given)

os.path.dirname(path) - It is used to return the directory name from the path given

os.path.isabs(path) - It specifies whether the path is absolute or not

os.path.isdir(path) - This function specifies whether the path is existing directory or not

os.path.isfile(path) - This function specifies whether the path is existing file or not

os.path.normcase(path) - This function normalizes the case of the pathname specified

os.path.normpath(path) - This function normalizes the path names by collapsing redundant separators and up-level references

#Example1

```
import  os
print(os.getcwd())          #prints the current working directory
os.mkdir("tmp")             #creates the directory tmp in the cwd
os.chdir("tmp")             #moves to the directory tmp
os.chdir("..")
os.rmdir("tmp")
print(len(os.listdir()))
```

#Example2

```
import os.path as p
print(p.basename("/home/user/sample.txt"))      → sample.txt
print(p.dirname("/home/user/sample.txt"))       → /home/user
print(p.isabs("/home/user/"))                   → True
print(p.isdir("/home/user/sample.txt"))         → False
print(p.isfile("/home/user/sample.txt"))        → True
```