
New
syllabus
2020-21
Chapter 3
File Handling
Computer Science
Class XII (As per CBSE Board)

Need for a data file

- To Store data in organized manner
- To store data permanently
- To access data faster
- To Search data faster
- To easily modify data later on

File Handling

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.

File Handling

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.e.g. .txt,.rtf**,.csv** etc.

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.e.g. .bmp,.cdr etc.

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

File Handling

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 resources. 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

open Function-

Before any reading or writing operation of any file,it must be opened first
of all.Python provide built in function open() for it.On calling of this function creates file object for file operations.

Syntax

file object = open(<file_name>, <access_mode>,< buffering>)

file_name = name of the file ,enclosed in double quotes.

access_mode= Determines the what kind of operations can be performed with file,like read,write etc.

Buffering = for no buffering set it to 0.for line buffering set it to 1.if it is greater than 1 ,then it is buffer size.if it is negative then buffer size is system default.

File Handling

File opening modes-

Sr. No.	Mode & Description
1	r - reading only. Sets the 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 existing file if the file exists. If not creates a new file for r & w.
8	**wb+** - same as w+ mode but with binary file.
9	a - for appending. Moves the pointer at end of file. Creates new file for writing if not exist.
10	**ab** – same as a but with binary file.
11	a+ - for both appending and reading. Moves file pointer at end. If file does not exist, creates new file for reading and writing.
12	**ab+** - same as a+ mode but with binary mode.

File Handling

Basic Text file operations

- Open (filename – absolute or relative path, mode)
- Close a text file
- Reading/Writing data
- Manipulation of data
- Appending data into a text file

File Handling

Methods of os module

Before Starting file operation following methods must be learn by a programmer to perform file system related methods available in os module(standard module) which can be used during file operations.

1. The rename() method used to rename the file.
Syntax os.rename(current_file_name, new_file_name)
2.The remove() method to delete file.
Syntax os.remove(file_name)
3.The mkdir() method of the os module to create directories in the current directory.
Syntax os.mkdir("newdir")
4.The *chdir()* method to change the current directory. Syntax os.chdir("newdir")
5.The getcwd() method displays the current directory.
Syntax os.getcwd()
6. The rmdir() method deletes the directory.
Syntax os.rmdir('dirname')

e.g.program

```
import os
print(os.getcwd())
os.mkdir("newdir")
os.chdir("newdir")
print(os.getcwd())
```

File Handling

Absolute Path vs Relative Path

One must be familiar with absolute & relative path before starting file related operations.

The absolute path is the full path to some place on your computer. The relative path is the path to some file with respect to your current working directory (PWD). For example:

Absolute path: C:/users/admin/docs/staff.txt

If PWD is C:/users/admin/, then the relative path to staff.txt would be: docs/staff.txt

Note, PWD + relative path = absolute path.

Cool, awesome. Now, if we write some scripts which check if a file exists.

```
os.chdir("C:/users/admin/docs")
os.path.exists("staff.txt")
```

This returns TRUE if stuff.txt exists and it works.

Now, instead if we write,

```
os.path.exists("C:/users/admin/docs/staff.txt")
```

This will returns TRUE.

If we don't know where the user executing the script from, it is best to compute the absolute path on the user's system using os and __file__.

__file__ is a global variable set on every Python script that returns the relative path to the *.py file that contains it.

e.g.program

```
import os
print(os.getcwd())
os.mkdir("newdir1")
os.chdir("newdir1")
print(os.getcwd())
my_absolute_dirpath =
os.path.abspath(os.path.dirname(__file__))
print(my_absolute_dirpath)
```

File Handling

File object attributes / 1. open a text file

- closed: It returns true if the file is closed and false when the file is open.
- encoding: Encoding used for byte string conversion.
- mode: Returns file opening mode
- name: Returns the name of the file which file object holds.
- newlines: Returns “\r”, “\n”, “\r\n”, None or a tuple containing all the newline types seen.

E.g. Program

#1 open text file

```
f = open("a.txt", 'a+')
print(f.closed)
print(f.encoding)
print(f.mode)
print(f.newlines)
print(f.name)
```

OUTPUT

```
False
cp1252
a+
None
a.txt
```

File Handling

2. Close a text file

close(): Used to close an open file. After using this method,an opened file will be closed and a closed file cannot be read or written any more.

E.g. program

```
f = open("a.txt", 'a+')
print(f.closed)
print("Name of the file is",f.name)
f.close()                    #2 close text file
print(f.closed)
```

OUTPUT

```
False
Name of the file is a.txt
True
```

File Handling

3. Read/write text file

The write() Method

It writes the contents to the file in the form of string. It does not return value. Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.

The read() Method

It reads the entire file and returns it contents in the form of a string. Reads at most size bytes or less if end of file occurs.if size not mentioned then read the entire file contents.

File Handling

Read/Write a Text file

write() ,read() Method based program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

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

OUTPUT

```
Welcome to python.mykvs.in
Regularly visit python.mykvs.in
```

Note : for text file operation file extension should be .txt and opening mode without 'b' & text file handling relevant methods should be used for file operations.

File Handling

Read Text file

readline([size]) method: Read no of characters from file if size is mentioned till eof.read line till new line character.returns empty string on EOF.

e.g. program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
text = f.readline()
print(text)
text = f.readline()
print(text)
f.close()
```

OUTPUT

```
Welcome to python.mykvs.in
Regularly visit python.mykvs.in
```

File Handling

Read Text file

readlines([size]) method: Read no of lines from file if size is mentioned or all contents if size is not mentioned.

e.g.program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
text = f.readlines(1)
print(text)
f.close()
```

OUTPUT

```
['Welcome to python.mykvs.in\n']
```

NOTE – READ ONLY ONE LINE IN ABOVE PROGRAM.

File Handling

Read a Text File

Iterating over lines in a file
e.g.program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
for text in f.readlines():
    print(text)
f.close()
```

File Handling

Read a Text file

Processing Every Word in a File

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
for text in f.readlines():
    for word in text.split( ):
        print(word)
f.close()
```

OUTPUT

```
Welcome
to
python.mykvs.in
Regularly
visit
python.mykvs.in
```

File Handling

Getting & Resetting the Files Position

The *tell()* method of python tells us the current position within the file,where as The *seek(offset[, from])* method changes the current file position. If *from* is 0, the beginning of the file to seek. If it is set to 1, the current position is used . If it is set to 2 then the end of the file would be taken as seek position. The offset argument indicates the number of bytes to be moved.

e.g.program

```
f = open("a.txt", 'w')
line = 'Welcome to python.mykvs.in\nRegularly visit python.mykvs.in'
f.write(line)
f.close()

f = open("a.txt", 'rb+')
print(f.tell())
print(f.read(7)) # read seven characters
print(f.tell())
print(f.read())
print(f.tell())
f.seek(9,0) # moves to 9 position from begining
print(f.read(5))
f.seek(4, 1) # moves to 4 position from current location
print(f.read(5))
f.seek(-5, 2) # Go to the 5th byte before the end
print(f.read(5))
f.close()
```

OUTPUT

```
0
b'Welcome'
7
b' to
python.mykvs.in\r\nRe
gularly visit
python.mykvs.in'
59
b'o pyt'
b'mykvs'
b'vs.in'
```

File Handling

4. Modify a Text file

There is no way to insert into the middle of a file without re-writing it. We can append to a file or overwrite part of it using seek but if we want to add text at the beginning or the middle, we'll have to rewrite it.

It is an operating system task, not a Python task. It is the same in all languages.

What we usually do for modification is read from the file, make the modifications and write it out to a new file called temp.txt or something like that. This is better than reading the whole file into memory because the file may be too large for that. Once the temporary file is completed, rename it the same as the original file. This is a good, safe way to do it because if the file write crashes or aborts for any reason in between, we still have our untouched original file.

File Handling

Modify a Text file

Replace string in the same File

```
fin = open("dummy.txt", "rt")
data = fin.read()
data = data.replace('my', 'your')
fin.close()

fin = open("dummy.txt", "wt")
fin.write(data)
fin.close()
```

What have we done here?

Open file dummy.txt in read text mode rt.

fin.read() reads whole text in dummy.txt to the variable data.

data.replace() replaces all the occurrences of my with your in the whole text.

fin.close() closes the input file dummy.txt.

In the last three lines, we are opening dummy.txt in write text wt mode and writing the data to dummy.txt in replace mode. Finally closing the file dummy.txt.

Note :- above program is suitable for file with small size.

File Handling

Modify a Text file

Replace string using temporary file

```
import os
f=open("d:\\a.txt","r")
g=open("d:\\c.txt","w")
for text in f.readlines():
    text=text.replace('my','your')
    g.write(text)
f.close()
g.close()
os.remove("d:\\a.txt")
os.rename("d:\\c.txt","d:\\a.txt")
print("file contents modified")
```

Note- above program is suitable for large file.Here line by line is being read from a.txt file in text string and text string been replaced ‘my’ with ‘your’ through replace() method.Each text is written in c.txt file then close both files ,remove old file a.txt through remove method of os module and rename c.txt(temporary file) file with a.txt file.After execution of above program all occurances of ‘my’ will be replaced with ‘your’.For testing of above program create a.txt file in d: drive with some substring as ‘my’.Run above program and check changes.

File Handling

5. Append content to a Text file

```
f = open("a.txt", 'w')
line = 'Welcome to python.mykvs.in\nRegularly visit python.mykvs.in'
f.write(line)
f.close()

f = open("a.txt", 'a+')
f.write("\nthanks")
f.close()

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

APPEND CODE

OUTPUT

```
Welcome to python.mykvs.in
Regularly visit python.mykvs.in
thanks
```

File Handling

Standard input, output, and error streams in python

Most programs make output to "standard out“,input from "standard in“, and error messages go to standard error).standard output is to monitor and standard input is from keyboard.

e.g.program

```
import sys
a = sys.stdin.readline()
sys.stdout.write(a)
a = sys.stdin.read(5)#entered 10 characters.a contains 5 characters.
#The remaining characters are waiting to be read.
sys.stdout.write(a)
b = sys.stdin.read(5)
sys.stdout.write(b)
sys.stderr.write("\ncustom error message")
```

File Handling

Write and read a Binary file

#e.g. program

```
binary_file=open("D:\\binary_file.dat",mode="wb+")
text="Hello 123"
encoded=text.encode("utf-8")
binary_file.write(encoded)
binary_file.seek(0)
binary_data=binary_file.read()
print("binary:",binary_data)
text=binary_data.decode("utf-8")
print("Decoded data:",text)
```

Note :- Opening and closing of binary file is same as text file opening and closing.While opening any binary file we have to specify ‘b’ in file opening mode.

In above program binary_file.dat is opened in wb+ mode so that after writing ,reading operation can be done on binary file.String variable text hold text to be encoded before writing with the help of encode method().utf-8 is encoding scheme.after writing text ,we again set reading pointer at beginning with the help of seek() method.then read the text from file and decode it with the help of decode() method then display the text.

File Handling

Binary file Operation using pickle module

The problem with the approach of previous slide comes from the fact that it is not very easy to use when we want to write several objects into the binary file. For instance, consider what we would have to do if we wished to write string, integer and perhaps even the contents of a list or dictionary into the file. How would we read the contents of this file later? This is not a trivial task, especially if some of the objects can have variable lengths.

Fortunately, Python has a module which does this work for us and is extremely easy to use. This module is called pickle; it provides us with the ability to serialize and deserialize objects, i.e., to convert objects into bitstreams which can be stored into files and later be used to reconstruct the original objects.

File Handling

Binary file Operation using pickle module

pickle.dump() function is used to store the object data to the file. It takes 3 arguments.First argument is the object that we want to store. The second argument is the file object we get by opening the desired file in write-binary (wb) mode. And the third argument is the key-value argument. This argument defines the protocol. There are two type of protocol – pickle.HIGHEST_PROTOCOL and pickle.DEFAULT_PROTOCOL.

Pickle.load() function is used to retrieve pickled data.The steps are quite simple. We have to use pickle.load() function to do that. The primary argument of pickle load function is the file object that you get by opening the file in read-binary (rb) mode.

File Handling

Binary file R/W Operation using pickle module

```
import pickle
output_file = open("d:\\a.bin", "wb")
myint = 42
mystring = "Python.mykvs.in!"
mylist = ["python", "sql", "mysql"]
mydict = { "name": "ABC", "job": "XYZ" }
pickle.dump(myint, output_file)
pickle.dump(mystring, output_file)
pickle.dump(mylist, output_file)
pickle.dump(mydict, output_file)
output_file.close()
input_file = open("d:\\a.bin", "rb")
myint = pickle.load(input_file)
mystring = pickle.load(input_file)
mylist = pickle.load(input_file)
mydict = pickle.load(input_file)
print("myint = %s" % myint)
print("mystring = %s" % mystring)
print("mylist = %s" % mylist)
print("mydict = %s" % mydict)
input_file.close()
```

In this program we open a.bin file in binary mode using ‘wb’ specification and create and store value in 4 different data objects i.e. int,string,list,dict.Write these into binary file using pickle.dump() method then close this file and open the same for reading operation.We read the content using load method() and display the value read from file.

To use dump and load we have to import pickle module first of all.

File Handling

Iteration over Binary file - pickle module

```
import pickle
output_file = open("d:\\a.bin", "wb")
myint = 42
mystring = "Python.mykvs.in!"
mylist = ["python", "sql", "mysql"]
mydict = { "name": "ABC", "job": "XYZ" }
pickle.dump(myint, output_file)
pickle.dump(mystring, output_file)
pickle.dump(mylist, output_file)
pickle.dump(mydict, output_file)
output_file.close()
with open("d:\\a.bin", "rb") as f:
    while True:
        try:
            r=pickle.load(f)
            print(r)
            print("Next data")
        except EOFError:
            break
f.close()
```

Read objects one by one

File Handling

Insert/append record in a Binary file - pickle module

```
rollno = int(input('Enter roll number:'))
name = input('Enter Name:')
marks = int(input('Enter Marks'))

#Creating the dictionary
rec = {'Rollno':rollno,'Name':name,'Marks':marks}

#Writing the Dictionary
f = open('d:/student.dat','ab')
pickle.dump(rec,f)
f.close()
```

Here we are creating dictionary rec to dump it in student.dat file

File Handling

Read records from a Binary file - pickle module

```
f = open('d:/student.dat','rb')
    while True:
        try:
            rec = pickle.load(f)
            print('Roll Num:',rec['Rollno'])
            print('Name:',rec['Name'])
            print('Marks:',rec['Marks'])
        except EOFError:
            break
    f.close()
```

Here we will iterate using infinite while loop and exit on end of file is reached.at each iteration a dictionary data is read into rec and then values are being displayed

File Handling

Search record in a Binary file - pickle module

```
f = open('d:/student.dat','rb')
    flag = False
    r=int(input("Enter rollno to be searched"))
    while True:
        try:
            rec = pickle.load(f)
            if rec['Rollno'] == r:
                print('Roll Num:',rec['Rollno'])
                print('Name:',rec['Name'])
                print('Marks:',rec['Marks'])
                flag = True
        except EOFError:
            break
    if flag == False:
        print('No Records found')
    f.close()
```

Here value of r to be searched will be compared with rollno value of file in each iteration/next record and if matches then relevant data will be shown and flag will be set to True otherwise it will remain False and 'No Record Found message will be displayed'

File Handling

Update record of a Binary file - pickle module

```
f = open('d:/student.dat','rb')
    reclst = []
    r=int(input("enter roll no to be updated"))
    m=int(input("enter correct marks"))
    while True:
        try:
            rec = pickle.load(f)
            reclst.append(rec)
        except EOFError:
            break
    f.close()
    for i in range (len(reclst)):
        if reclst[i]['Rollno']==r:
            reclst[i]['Marks'] = m
    f = open('d:/student.dat','wb')
    for x in reclst:
        pickle.dump(x,f)
    f.close()
```

Here we are reading all records from binary file and storing those in reclst list then update relevant roll no /marks data in reclst list and create /replace student.dat file with all data of reclst.

If large data are in student.dat file then It's alternative way can be using temporary file creation with corrected data then using os module for remove and rename method (just similar to modification of text file)

File Handling

Delete record of a Binary file - pickle module

```
f = open('d:/student.dat','rb')
    reclst = []
    r=int(input("enter roll no to be deleted"))
    while True:
        try:
            rec = pickle.load(f)
            reclst.append(rec)
        except EOFError:
            break
    f.close()
    f = open('d:/student.dat','wb')
    for x in reclst:
        if x['Rollno']==r:
            continue
        pickle.dump(x,f)
    f.close()
```

Here we are reading all records from binary file and storing those in reclst list then write all records except matching roll no(with the help of continue statement) in student.dat file.Due to wb mode old data will be removed.

If large data are in student.dat file then It's alternative way can be using temporary file creation with corrected data then using os module for remove and rename method (just similar to deletion from text file)

File Handling

Download python file/program for interactive binary data file handling (includes insert/search/update/delete opeartion)

File Handling

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

File Handling

CSV FILE

(Comma separated value)

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)

File Handling

Write / Read 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()
```

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.