BINARY FILE HANDLING IN PYTHON

Binary File

- In binary file data is in unreadable format and to work on binary file we have to convert the data into readable form for read as well as write operation.
- The binary file store some objects which have some structure associated with them. Like list, nested list, tuple, dictionary etc.
- These objects first serialized and then store in binary file.
- If a binary file (existing) open for reading purpose, when last record is reached (EOF reached), it may raise an EOFError exception, if not handled properly.
- Thus a binary file should open for reading purpose either in "try and except" blocks or using "with statement".

Serialization or Pickling:

It is the process of transforming a python object to a stream of bytes called byte streams. These byte streams can then be stored in binary file. Serialization process is also called pickling.

De-serialization or un-pickling:

It is the inverse of pickling process where a byte stream is converted back to Python object.

The pickle module implements the algorithm for serializing and de-sterilizing the python objects and deals with binary files. The Pickle Module must be imported to read and write object in binary file.

MODES OF BINARY FILES

- b = Open the binary file
- rb = Open binary file in read only mode (file must be existed)
- wb = Open binary file in write only mode (new file created, over written if existed)
- ab = Open binary file in append (write only) mode. (open if existed otherwise create new file)
- rb+ = Open binary file in read & write mode. (file must be existed)
- wb+ = Open binary file in write & read mode. (new file created, over written if existed)
- ab+ = Open binary file in append (write & read) mode. (open if existed otherwise create new file)

BASIC OPERATIONS IN BINARY FILE

- Creating a new file
- Reading from file
- Writing into file
- Appending the file
- Searing in File
- Deleting data from file
- Creating a copy of file

OPEN AND CLOSE BINARY FILE

open( )

Syntax:

File_handler/File_Object=open(file_name, access mode)

Example:

file_obj= open("bin_file.dat",'wb')

This statement opens bin_file.dat binary in write mode
Note : if file mode is not mentioned in open function then default file mode i.e 'rb' is used,

close() : The close( ) method of a file object flushes any unwritten information and close the file object after which no more writing can be done.

Example

file_obj.close( )

WRITING ONTO FILE- PICKLING

To write an object into file, the file should be opened in write mode. The dump( ) function of pickle module used to write object into file.

Syntax:

```
import pickle
File_Handler=open("Bin_file.dat",'wb')
pickle.dump(python_Object_to_be_Written, File_Handler)
```

Expamle:

```
# write Dictionary Object in Binary file
import pickle
stud={}
fwb=open("student.dat","wb")
choice='y'
while choice.lower()=='y':
  rno=int(input("Enter Roll No: "))
  name=input("Enter Name: ")
  marks=float(input("Enter Marks(out of 500): "))
  per=marks/5
  if(per>=33):
    res="Pass"
  else:
    res="Fail"
  stud['rollno']=rno
  stud['name']=name
  stud['Marks']=marks
  stud['percent']=per
  stud['result']=res
  pickle.dump(stud, fwb)
  print("Record Saved in File")
  choice=input("Want to Enter More Record(Y/N)? ")
```

```
fwb.close()
```

APPENDING RECORD IN BINARY FILE

Binary file must open in append mode (i.e., "ab'") to append the records in file. A file opened in append mode will retain the previous records and append the new records written in the file.

Syntax:

```
import pickle
File_Handler=open(“Bin_file.dat”,’ab’)
pickle.dump(python_Object_to_be_Written, File_Handler)
```

Expamle:

```
# Append the Binary file
import pickle
stud={}
fwb=open("student.dat","ab")
choice='y'
while choice.lower()=='y':
   rno=int(input("Enter Roll No: "))
   name=input("Enter Name: ")
   marks=float(input("Enter Marks(out of 500): "))
   per=marks/5
   if(per>=33):
      res="Pass"
   else:
      res="Fail"
   stud['rollno']=rno
   stud['name']=name
   stud['Marks']=marks
   stud['percent']=per
   stud['result']=res
   pickle.dump(stud, fwb)
   print("Record Saved in File")
   choice=input("Want to Enter More Record(Y/N)? ")
fwb.close()
```

READING FROM BINARY FILE: UN-PICKLING

While working with binary file for reading purpose the runtime exception EOFError raised when it encounter the EOF position. This EOFError exception can handle with two ways.

1. Use of try and except block

The try and except statements together, can handle runtime exceptions. In the try block, i.e., between the try and except keywords, write the code that can generate an exception and in the except block, ie., below the except keyword, write what to do when the exception (EOF - end of file) has occurred

File_object=("binary File Name",'access mode')

try:

……..

Write actual code work on binary file

……..

except EOFError:

……..

Write that Code here that can handle the error raised in try block.

……..

2. Use of with statement

The with statement is a compact statement which combines the opening of file, processing of file along with inbuilt exception handling and also close the file automatically after with block is over. Explicitly, we need not to mention any exception for the "with statement".

Example:

with open(" file name" , 'Access mode') as file_handler:

……

Write actual code work on binary file

```
# Read Records from Binary file

import pickle

stud={}

frb=open("student.dat","rb")

try:

  while True:

    stud=pickle.load(frb)
```

```
            print(stud)
        except EOFError:
            frb.close()
```

SEARCHING RECORD FROM FILE

```
# Searching Data
import pickle
stud={}
found=0
print("searching in file student.dat...")
try:
    frb=open("student.dat", "rb")
    while True:
        stud=pickle.load(frb)
        if stud['percent']>51.0:
            print(stud)
            found+=1
except EOFError:
    if found==0:
        print("No Record found with marks>81")
        frb.close()
    else:
        print(found," Record(s) Searched")
        frb.close()
```

COPY OF FILE IN ANOTHER FILE

```
#CREATIN A COPY OF EXISTING FILE PROGRAM
```

```
import pickle
def fileCopy():
    ifile = open("student.dat","rb")
    ofile = open("newfile.dat","wb")
    try:
        while True:
            rec=pickle.load(ifile)
            pickle.dump(rec,ofile)
    except EOFError:
        ifile.close()
        ofile.close()
        print("Copied successfully")

def display1():
    ifile = open("student.dat","rb")
    print("----Records of Main file---")
    try:
        while True:
            rec=pickle.load(ifile)
            print(rec)
    except EOFError:
        ifile.close()
def display2():
    ofile = open("newfile.dat","rb")
    print("----Records of Copy file---")
    try:
```

```
        while True:
            rec=pickle.load(ofile)
            print(rec)
        except EOFError:
            ofile.close()


fileCopy()

display1()

display2()
```

RANDOM ACCESS & UPDATING BINARY FILE

Python provides two functions that help to manipulate the position of file pointer and we can read and write from desired position in the file. The Two functions of Python are: tell() and seek()

(i) The tell() Function

tell() method can be used to get the current position of File Handle in the file. This method takes no parameters and returns an integer value. Initially file pointer points to the beginning of the file(if not opened in append mode).

Syntax

fileObject.tell( )

This method returns the current position of the file read/write pointer within the file.

(ii) The seek( ) Fuction

In Python, seek() function is used to change the position of the File Handle to a given specific position in the file.

Syntax

fileObject.seek(offset, mode)

where

offset is a number-of-bytes

mode is a number from 0 or 1 or 2

0: sets the reference point at the beginning of the file.

1: sets the reference point at the current file position.

2: sets the reference point at the end of the file.

Example:

```
F=open("chapter.txt,'r')

f.seek(20)    # will place the file pointer at 20th byte from the beginning of the file (default)
```

f.seek(20,1) # wil place the file pointer at 20th byte ahead of current file-pointer position (mode = 1)

f.seek(-20,2) # will place file pointer at 20 bytes behind (backward direction) from end-of file (mode = 2)

f.seek(-10,1) # will place file pointer at 5 bytes behind from current file-pointer position (mode = 1)

Note:

Backward movement of file-pointer is not possible from the beginning of the file (BOF).

Forward movement of file-pointer is not possible from the end of file (EOF).

```
import pickle
stu={ }
found =false
# open binary file in read and write mode
Rfile=open("stu.dat", "rb+")

# Read from the file
Try:
    While True:
        Pos=rfile.tell()
        Stu=pickle.load(rfile)
        If stu["marks"] > 81:
            Stu["Marks"] +=2
            Rfile.seek(pos)
            Pickle.dump(stu, rfile)
            Found=True
Excption EOFError:
    If founf =False:
        Print("Sorry, No Record found")
    Else:
        Print("Record(s) successfully updated")
Rfile.close()
```

DELETE DATA FROM FILE

Python Delete File

To delete a file, you must import the OS module, and run its os.remove() function:

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:

```
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:

```
import os
os.rmdir("myfolder")
```

Note: You can only remove *empty* folders.

PROGRAM

```
import pickle
import os
def delete_rec():
   f1 = open("stu.dat","rb")
   f2 = open("temp.dat","wb")
   s=int(input("Enter rollno to delete:"))
   try:
      while True:
         d = pickle.load(f1)
         if d["rollno"]!=s:
            pickle.dump(d,f2)
   except EOFError:
      print("Record Deleted.")
   f1.close()
   f2.close()
   os.remove("stu.dat")
   os.rename("temp.dat","sales.dat")
delete_rec()
```