File Handling

As the part of programming requirement, we have to store our data permanently for future purpose. For this requirement we should go for files.

Files are very common permanent storage areas to store our data.

Types of Files:

There are 2 types of files

1. Text Files:

Usually we can use text files to store character data
eg: abc.txt

2. Binary Files:

Usually we can use binary files to store binary data like images,video files, audio files etc...

Opening a File:

Before performing any operation (like read or write) on the file,first we have to open that file.For this we should use Python's inbuilt function open()

But at the time of open, we have to specify mode,which represents the purpose of opening file.

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

The allowed modes in Python are

1. r → open an existing file for read operation. The file pointer is positioned at the beginning of the file.If the specified file does not exist then we will get FileNotFoundError.This is default mode.

2. w → open an existing file for write operation. If the file already contains some data then it will be overridden. If the specified file is not already avaialble then this mode will create that file.

3. a → open an existing file for append operation. It won't override existing data.If the specified file is not already avaialble then this mode will create a new file.

4. r+ → To read and write data into the file. The previous data in the file will not be deleted.The file pointer is placed at the beginning of the file.

5. w+ → To write and read data. It will override existing data.

6. a+ → To append and read data from the file.It wont override existing data.

7. x → To open a file in exclusive creation mode for write operation. If the file already exists then we will get FileExistsError.

Note: All the above modes are applicable for text files. If the above modes suffixed with 'b' then these represents for binary files.

Eg: rb,wb,ab,r+b,w+b,a+b,xb

```
f = open("abc.txt","w")
```

We are opening abc.txt file for writing data.

Closing a File:

After completing our operations on the file,it is highly recommended to close the file. For this we have to use close() function.

```
f.close()
```

Various properties of File Object:

Once we opend a file and we got file object,we can get various details related to that file by using its properties.

name → Name of opened file
mode → Mode in which the file is opened
closed → Returns boolean value indicates that file is closed or not
readable()→ Retruns boolean value indicates that whether file is readable or not
writable()→ Returns boolean value indicates that whether file is writable or not.

Eg:

```
f=open("abc.txt",'w')
print("File Name: ",f.name)
print("File Mode: ",f.mode)
print("Is File Readable:  ",f.readable())
print("Is File Writable:  ",f.writable())
print("Is File Closed : ",f.closed)
f.close()
print("Is File Closed : ",f.closed)


Output
D:\Python_classes>py test.py
File Name:  abc.txt
File Mode:  w
Is File Readable:   False
Is File Writable:   True
Is File Closed :  False
Is File Closed :  True
```

Writing data to text files:

We can write character data to the text files by using the following 2 methods.

write(str)
writelines(list of lines)

Eg:

```
f=open("abcd.txt",'w')
f.write("Durga\n")
f.write("Software\n")
f.write("Solutions\n")
print("Data written to the file successfully")
f.close()
```

abcd.txt:
Durga
Software
Solutions

Note: In the above program, data present in the file will be overridden everytime if we run the program. Instead of overriding if we want append operation then we should open the file as follows.

```
f = open("abcd.txt","a")
```

Eg 2:

```
f=open("abcd.txt",'w')
list=["sunny\n","bunny\n","vinny\n","chinny"]
f.writelines(list)
print("List of lines written to the file successfully")
f.close()
```

abcd.txt:

```
sunny
bunny
vinny
chinny
```

Note: while writing data by using write() methods, compulsory we have to provide line seperator(\n),otherwise total data should be written to a single line.

Reading Character Data from text files:

We can read character data from text file by using the following read methods.

read()→ To read total data from the file
read(n) → To read 'n' characters from the file
readline()→ To read only one line
readlines()→ To read all lines into a list

Eg 1: To read total data from the file

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

Output
sunny
bunny
chinny
vinny
```

Eg 2: To read only first 10 characters:

```
f=open("abc.txt",'r')
data=f.read(10)
print(data)
f.close()

```

```
Output
sunny
bunn
```

Eg 3: To read data line by line:

```
f=open("abc.txt",'r')
line1=f.readline()
print(line1,end='')
line2=f.readline()
print(line2,end='')
line3=f.readline()
print(line3,end='')
f.close()

Output
sunny
bunny
chinny
```

Eg 4: To read all lines into list:

```
f=open("abc.txt",'r')
lines=f.readlines()
for line in lines:
    print(line,end='')
f.close()

Output
sunny
bunny
chinny
vinny
```

Eg 5:

```
f=open("abc.txt","r")
print(f.read(3))
print(f.readline())
print(f.read(4))
print("Remaining data")
print(f.read())

Output
sun
ny

bunn
Remaining data
```

```
y
chinny
vinny
```

The with statement:

The with statement can be used while opening a file.We can use this to group file operation statements within a block.
The advantage of with statement is it will take care closing of file,after completing all operations automatically even in the case of exceptions also, and we are not required to close explicitly.
Eg:

```
with open("abc.txt","w") as f:
   f.write("Durga\n")
   f.write("Software\n")
   f.write("Solutions\n")
   print("Is File Closed: ",f.closed)
print("Is File Closed: ",f.closed)

Output
Is File Closed:  False
Is File Closed:  True
```

The seek() and tell() methods:

tell():

==>We can use tell() method to return current position of the cursor(file pointer) from beginning of the file. [can you plese telll current cursor position]
The position(index) of first character in files is zero just like string index.

Eg:

```
f=open("abc.txt","r")
print(f.tell())
print(f.read(2))
print(f.tell())
print(f.read(3))
print(f.tell())
```

abc.txt:

```
sunny
bunny
chinny
vinny
```

Output:

```
0
su
2
nny
5
```

seek():

We can use seek() method to move cursor(file pointer) to specified location.
[Can you please seek the cursor to a particular location]

`f.seek(offset, fromwhere)`

offset represents the number of positions

The allowed values for second attribute(from where) are
0---->From beginning of file(default value)
1---->From current position
2--->From end of the file

Note: Python 2 supports all 3 values but Python 3 supports only zero.

Eg:

```
data="All Students are STUPIDS"
f=open("abc.txt","w")
f.write(data)
with open("abc.txt","r+") as f:
    text=f.read()
    print(text)
    print("The Current Cursor Position: ",f.tell())
    f.seek(17)
    print("The Current Cursor Position: ",f.tell())
    f.write("GEMS!!!")
    f.seek(0)
    text=f.read()
    print("Data After Modification:")
    print(text)

Output

All Students are STUPIDS
The Current Cursor Position:  24
The Current Cursor Position:  17
Data After Modification:
```

```
All Students are GEMS!!!
```

How to check a particular file exists or not?

We can use os library to get information about files in our computer.
os module has path sub module,which contains isFile() function to check whether a particular file exists or not?
os.path.isfile(fname)

Q. Write a program to check whether the given file exists or not. If it is available then print its content?

```
import os,sys
fname=input("Enter File Name: ")
if os.path.isfile(fname):
   print("File exists:",fname)
   f=open(fname,"r")
else:
   print("File does not exist:",fname)
   sys.exit(0)
print("The content of file is:")
data=f.read()
print(data)

Output
D:\Python_classes>py test.py
Enter File Name: durga.txt
File does not exist: durga.txt

D:\Python_classes>py test.py
Enter File Name: abc.txt
File exists: abc.txt
The content of file is:
All Students are GEMS!!!
All Students are GEMS!!!
All Students are GEMS!!!
All Students are GEMS!!!
All Students are GEMS!!!
All Students are GEMS!!!
```

Note:
sys.exit(0) ===>To exit system without executing rest of the program.
argument represents status code . 0 means normal termination and it is the default value.

Q. Program to print the number of lines,words and characters present in the given file?

```
import os,sys
fname=input("Enter File Name: ")
if os.path.isfile(fname):
   print("File exists:",fname)
   f=open(fname,"r")
else:
   print("File does not exist:",fname)
   sys.exit(0)
lcount=wcount=ccount=0
for line in f:
   lcount=lcount+1
   ccount=ccount+len(line)
   words=line.split()
   wcount=wcount+len(words)
print("The number of Lines:",lcount)
print("The number of Words:",wcount)
print("The number of Characters:",ccount)

Output
D:\Python_classes>py test.py
Enter File Name: durga.txt
File does not exist: durga.txt

D:\Python_classes>py test.py
Enter File Name: abc.txt
File exists: abc.txt
The number of Lines: 6
The number of Words: 24
The number of Characters: 149
```

abc.txt:

```
All Students are GEMS!!!
All Students are GEMS!!!
All Students are GEMS!!!
All Students are GEMS!!!
All Students are GEMS!!!
All Students are GEMS!!!
```

Handling Binary Data:

It is very common requirement to read or write binary data like images,video files,audio files etc.

Q. Program to read image file and write to a new image file?

```
f1=open("rossum.jpg","rb")
f2=open("newpic.jpg","wb")
bytes=f1.read()
f2.write(bytes)
print("New Image is available with the name: newpic.jpg")
```

Handling csv files:

CSV==>Comma seperated values

As the part of programming,it is very common requirement to write and read data wrt csv files. Python provides csv module to handle csv files.

Writing data to csv file:

```
import csv
with open("emp.csv","w",newline='') as f:
  w=csv.writer(f)  # returns csv writer object
  w.writerow(["ENO","ENAME","ESAL","EADDR"])
  n=int(input("Enter Number of Employees:"))
  for i in range(n):
    eno=input("Enter Employee No:")
    ename=input("Enter Employee Name:")
    esal=input("Enter Employee Salary:")
    eaddr=input("Enter Employee Address:")
    w.writerow([eno,ename,esal,eaddr])
print("Total Employees data written to csv file successfully")
```

Note: Observe the difference with newline attribute and without

with open("emp.csv","w",newline='') as f:
with open("emp.csv","w") as f:

Note: If we are not using newline attribute then in the csv file blank lines will be included between data. To prevent these blank lines, newline attribute is required in Python-3,but in Python-2 just we can specify mode as 'wb' and we are not required to use newline attribute.

Reading Data from csv file:

```
import csv
f=open("emp.csv",'r')
r=csv.reader(f) #returns csv reader object
data=list(r)
#print(data)
for line in data:
    for word in line:
        print(word,"\t",end='')
    print()

Output
D:\Python_classes>py test.py
ENO   ENAME   ESAL   EADDR
100    Durga   1000    Hyd
200    Sachin  2000    Mumbai
300    Dhoni   3000    Ranchi
```

Zipping and Unzipping Files:

It is very common requirement to zip and unzip files.
The main advantages are:

1. To improve memory utilization
2. We can reduce transport time
3. We can improve performance.

To perform zip and unzip operations, Python contains one in-bulit module zip file.
This module contains a class : ZipFile

To create Zip file:

We have to create ZipFile class object with name of the zip file,mode and constant ZIP_DEFLATED. This constant represents we are creating zip file.

```
f = ZipFile("files.zip","w","ZIP_DEFLATED")
```

Once we create ZipFile object,we can add files by using write() method.

```
f.write(filename)
```

Eg:

```python
from zipfile import *
f=ZipFile("files.zip",'w',ZIP_DEFLATED)
f.write("file1.txt")
f.write("file2.txt")
f.write("file3.txt")
f.close()
print("files.zip file created successfully")
```

To perform unzip operation:

We have to create ZipFile object as follows

`f = ZipFile("files.zip","r",ZIP_STORED)`

ZIP_STORED represents unzip operation. This is default value and hence we are not required to specify.
Once we created ZipFile object for unzip operation,we can get all file names present in that zip file by using namelist() method.

`names = f.namelist()`

Eg:

```python
from zipfile import *
f=ZipFile("files.zip",'r',ZIP_STORED)
names=f.namelist()
for name in names:
    print(   "File Name: ",name)
    print("The Content of this  file is:")
    f1=open(name,'r')
    print(f1.read())
    print()
```

Working with Directories:

It is very common requirement to perform operations for directories like

1. To know current working directory
2. To create a new directory
3. To remove an existing directory
4. To rename a directory
5. To list contents of the directory
etc...

To perform these operations,Python provides inbuilt module os,which contains several functions to perform directory related operations.

Q1. To Know Current Working Directory:

```
import os
cwd=os.getcwd()
print("Current Working Directory:",cwd)
```

Q2. To create a sub directory in the current working directory:

```
import os
os.mkdir("mysub")
print("mysub directory created in cwd")
```

Q3. To create a sub directory in mysub directory:

```
 cwd
  |-mysub
     |-mysub2
```

```
import os
os.mkdir("mysub/mysub2")
print("mysub2  created inside mysub")
```

Note: Assume mysub already present in cwd.

Q4. To create multiple directories like sub1 in that sub2 in that sub3:

```
 import os
os.makedirs("sub1/sub2/sub3")
print("sub1 and in that sub2 and in that sub3 directories created")
```

Q5. To remove a directory:

```
import os
os.rmdir("mysub/mysub2")
print("mysub2 directory deleted")
```

Q6. To remove multiple directories in the path:

```
import os
os.removedirs("sub1/sub2/sub3")
print("All 3 directories sub1,sub2 and sub3 removed")
```

Q7. To rename a directory:

```
import os
os.rename("mysub","newdir")
print("mysub directory renamed to newdir")
```

Q8. To know contents of directory:

os module provides listdir() to list out the contents of the specified directory. It won't display the contents of sub directory.

Eg:

```
import os
print(os.listdir("."))

Output
D:\Python_classes>py test.py
['abc.py', 'abc.txt', 'abcd.txt', 'com', 'demo.py', 'durgamath.py', 'emp.csv', '
file1.txt', 'file2.txt', 'file3.txt', 'files.zip', 'log.txt', 'module1.py', 'myl
og.txt', 'newdir', 'newpic.jpg', 'pack1', 'rossum.jpg', 'test.py', '__pycache__'
]
```

The above program display contents of current working directory but not contents of sub directories.

If we want the contents of a directory including sub directories then we should go for walk() function.

Q9. To know contents of directory including sub directories:

We have to use walk() function
[Can you please walk in the directory so that we can aware all contents of that directory]

os.walk(path,topdown=True,onerror=None,followlinks=False)

It returns an Iterator object whose contents can be displayed by using for loop

path-->Directory path. cwd means .
topdown=True --->Travel from top to bottom
onerror=None --->on error detected which function has to execute.
followlinks=True -->To visit directories pointed by symbolic links

Eg: To display all contents of Current working directory including sub directories:

```
import os
for dirpath,dirnames,filenames  in os.walk('.'):
   print("Current Directory Path:",dirpath)
   print("Directories:",dirnames)
   print("Files:",filenames)
   print()


 Output
 Current Directory Path: .
Directories: ['com', 'newdir', 'pack1', '__pycache__']
Files: ['abc.txt', 'abcd.txt', 'demo.py', 'durgamath.py', 'emp.csv', 'file1.txt'
, 'file2.txt', 'file3.txt', 'files.zip', 'log.txt', 'module1.py', 'mylog.txt', '
newpic.jpg', 'rossum.jpg', 'test.py']

Current Directory Path: .\com
Directories: ['durgasoft', '__pycache__']
Files: ['module1.py', '__init__.py']

...
```

Note: To display contents of particular directory,we have to provide that directory name as argument to walk() function.

os.walk("directoryname")

Q. What is the difference between listdir() and walk() functions?

In the case of listdir(), we will get contents of specified directory but not sub directory contents. But in the case of walk() function we will get contents of specified directory and its sub directories also.

Running Other programs from Python program:

os module contains system() function to run programs and commands.
It is exactly same as system() function in C language.

os.system("commad string")
  The argument is any command which is executing from DOS.

Eg:

```
import os
os.system("dir *.py")
os.system("py abc.py")
```

How to get information about a File:

We can get statistics of a file like size, last accessed time,last modified time etc by using stat() function of os module.

```
stats = os.stat("abc.txt")
```

The statistics of a file includes the following parameters:

st_mode==>Protection Bits
st_ino==>Inode number
st_dev===>device
st_nlink===>no of hard links
st_uid===>userid of owner
st_gid==>group id of owner
st_size===>size of file in bytes
st_atime==>Time of most recent access
st_mtime==>Time of Most recent modification
st_ctime==> Time of Most recent meta data change

Note:

st_atime, st_mtime and st_ctime returns the time as number of milli seconds since Jan 1st 1970 ,12:00AM. By using datetime module fromtimestamp() function,we can get exact date and time.

Q. To print all statistics of file abc.txt:

```
import os
stats=os.stat("abc.txt")
print(stats)

Output
os.stat_result(st_mode=33206, st_ino=844424930132788, st_dev=2657980798, st_nlin
k=1, st_uid=0, st_gid=0, st_size=22410, st_atime=1505451446, st_mtime=1505538999
, st_ctime=1505451446)
```

Q. To print specified properties:

```
import os
from  datetime  import *
stats=os.stat("abc.txt")
print("File Size in Bytes:",stats.st_size)
print("File Last Accessed Time:",datetime.fromtimestamp(stats.st_atime))
print("File Last Modified Time:",datetime.fromtimestamp(stats.st_mtime))

```

```
Output
File Size in Bytes: 22410
File Last Accessed Time: 2017-09-15 10:27:26.599490
File Last Modified Time: 2017-09-16 10:46:39.245394
```

Pickling and Unpickling of Objects:

Sometimes we have to write total state of object to the file and we have to read total object from the file.

The process of writing state of object to the file is called pickling and the process of reading state of an object from the file is called unpickling.

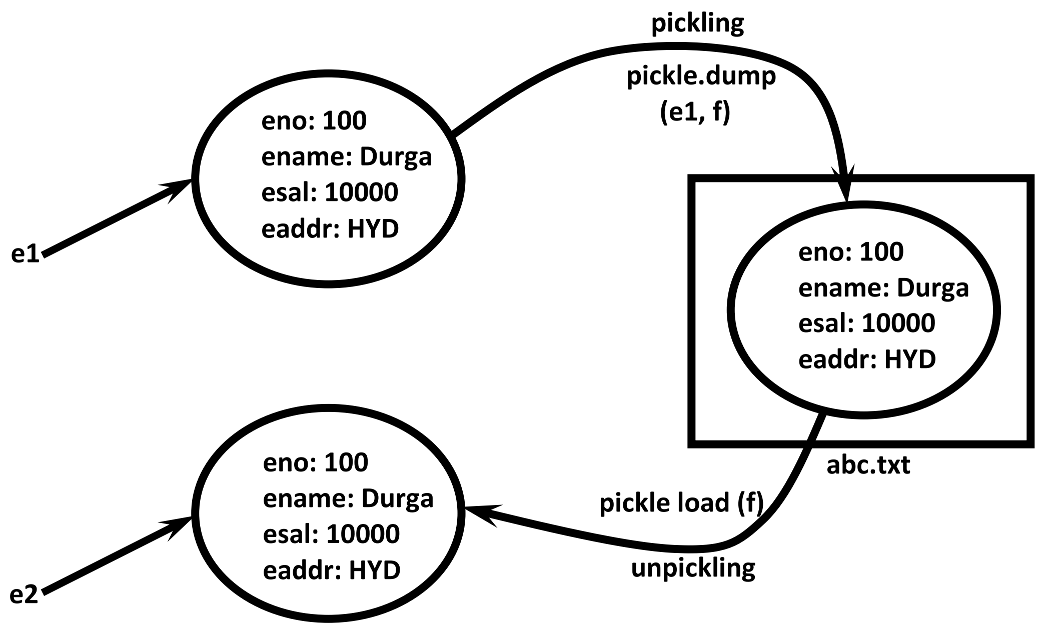
We can implement pickling and unpickling by using pickle module of Python.

pickle module contains dump() function to perform pickling.

```
pickle.dump(object,file)
```

pickle module contains load() function to perform unpickling

```
obj=pickle.load(file)
```



Writing and Reading State of object by using pickle Module:

```
import pickle
class Employee:
   def __init__(self,eno,ename,esal,eaddr):
      self.eno=eno;
      self.ename=ename;
      self.esal=esal;
      self.eaddr=eaddr;
   def display(self):
      print(self.eno,"\t",self.ename,"\t",self.esal,"\t",self.eaddr)
with open("emp.dat","wb") as f:
   e=Employee(100,"Durga",1000,"Hyd")
   pickle.dump(e,f)
   print("Pickling of Employee Object completed...")

with open("emp.dat","rb") as f:
   obj=pickle.load(f)
   print("Printing Employee Information after unpickling")
   obj.display()
```

Writing Multiple Employee Objects to the file:

emp.py:

```
class Employee:
   def __init__(self,eno,ename,esal,eaddr):
      self.eno=eno;
      self.ename=ename;
      self.esal=esal;
      self.eaddr=eaddr;
   def display(self):

   print(self.eno,"\t",self.ename,"\t",self.esal,"\t",self.eaddr)
```

pick.py:

```
import emp,pickle
f=open("emp.dat","wb")
n=int(input("Enter The number of Employees:"))
for i in range(n):
   eno=int(input("Enter Employee Number:"))
   ename=input("Enter Employee Name:")
   esal=float(input("Enter Employee Salary:"))
   eaddr=input("Enter Employee Address:")
   e=emp.Employee(eno,ename,esal,eaddr)
   pickle.dump(e,f)
print("Employee Objects pickled successfully")
```

unpick.py:

```python
import emp,pickle
f=open("emp.dat","rb")
print("Employee Details:")
while True:
    try:
        obj=pickle.load(f)
        obj.display()
    except EOFError:
        print("All employees Completed")
        break
f.close()
```