File Handling Date18/12/2020 Time:11am to 12pm

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 available 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 available then this mode will create a new file.

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

Example:

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.

Syntax:

f.close();

- **Various properties of File Object:**

Once we opened 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.

Program

```
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)
```

- Writing data to text files:

We can write character data to the text files by using the following 2 methods.
1. write(str)
2. writelines(list of lines)

Example:

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

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")
```

Example2:

```
f=open("abcd.txt",'w')
list=["abc\n","bbb\n","ccc\n","nnn"]
f.writelines(list)
print("List of lines written to the file successfully")
f.close()
```

- **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

Example1:
1: To read total data from the file

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

Example 2:
To read only first 10 characters:

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

Example 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()
```

Example 4:
To read all lines into list:

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

Date:21/12/2020 Time:12:15pm to 1:30pm

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.

Example:

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

Binary Files:

In this file we can store data in binary format.
Execution is fast as compared to text file.
Data can be stored like image file,video,audio file etc.

Pickling and Unpickling of Objects:

Pickling:
By pickling we can store data in binary files by using python object.
Or
The process of writing state of object to the file is called pickling

Unpickling
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 data to Binary files:**

```
import pickle
f1=open("xyz.dat","wb")
pickle.dump("Hello World",f1)
l1=["Mini",15,]
pickle.dump(l1,f1)
print("Data write successfully")
f1.close()
```

- **Reading data from binary file**

```
f2=open("xyz.dat","rb")
k=pickle.load(f2)
print(k)
k=pickle.load(f2)
print(k)
f2.close();
```

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

```
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,"aaa",1000,"Pune")
        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()
```

Program of Writing Multiple Employee Objects to the file:

```
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)


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")
f.close()


f1=open("emp.dat","rb")
print("Employee Details:")
while True:
        try:
                obj=pickle.load(f1)
                obj.display()
        except EOFError:
                print("All employees Completed")
                break
f1.close()
```

Manipulating 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

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

1.To Know Current Working Directory:

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

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

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

3.To create a sub directory in mysub directory:

```
import os
os.mkdir("java/dotnet")
print("dotnet created inside java")
```

4.To remove a directory:

```
import os
os.rmdir("java/dotnet")
print("dotnet directory deleted")
```

5.To rename a directory:

```
import os
os.rename("java","python")
print("java directory renamed to python")
```

6.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.

Ex:

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

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.

7. 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]

Syntax:

```
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

Ex:

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

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

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.