File I/O Handling

Bhavika Vaghela

PICA - BCA

- What is file?
- Why it is required?
- In which memory your data is store while you run program?

Cont..

- Files are a way to save data permanently. Everything you've learned so far is resident only in memory; as soon as you close down Python or turn off your computer, it goes away.
- The files that Python creates are manipulated by the computer's file system. Python is able to use operating system specific functions to import, save, and modify files.
- In this section of the tutorial, we will learn all about file handling in python including, creating a file, opening a file, closing a file, writing and appending the file, etc.

Cont..

- Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.
- Python treats files as a data stream, i.e. each file is read and stored as a sequential flow of bytes. Each file has an end-of-file (EOF) marker denoting when the last byte of data has been read from it.

Cont..

Open File :

- Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object.
- If file is not there in your system than open() function will give an error. But if we open file in write mode using open() function than it will created the file.
- **Syntax:**

file object = open(<file-name>, <access mode>, <buffering>)

E.g. fileptr=open("myfile.txt","w")

Cont..

- **file_name** – The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode** – The access_mode determines the mode in which the file has to be opened, i.e., read, write, append etc..
- **buffering** – If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Cont..

- The files can be accessed using various modes like read, write, or append.

Access Mode	**Description**
r	It opens the file to read-only. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
rb	It opens the file to read only in binary format.
r+	It opens the file to read and write both.
rb+	It opens the file to read and write both in binary format.
w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists
wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists

Cont..

Access mode	Description
w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists.
wb+	It opens the file to write and read both in binary format.
a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists
ab	It opens the file in the append mode in binary format
a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists
ab+	It opens a file to append and read both in binary format.

Cont..

Simple example of file

```
#opens the file file.txt in read mode
fileptr = open("file.txt","r")

if fileptr:
    print("file is opened successfully")
```

- Here open() is function to open file.
- fileptr is object of file.
- file.txt is name of file.
- r is access mode of file.
- If condition is used to check value of file object if its true than print statement will be executed.

Cont..

The *file* Object Attributes

- Once a file is opened and you have one file object, you can get various information related to that file.
- Here is a list of all attributes related to file object.

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise. Softspace is read write attribute which used internally by print statement.

here file is object of file.

Cont..

Example of file attribute :

```
# Open a file

myfile = open(“mydata.txt", "wb")
print "Name of the file: ", myfile.name
print "Closed or not : ", myfile.closed
print "Opening mode : ", myfile.mode
print "Softspace flag : ", myfile.softspace



Output :

Name of the file: mydata.txt
Closed or not : False
Opening mode : wb
Softspace flag : 0
```

Cont..

Like open() function we have many more function to handle file.

close() method

- Once all the operations are done on the file, we must close it through our python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.
- Python automatically closes a file when the reference object of a file is reassigned to another file.
- It is a good practice to use the close() method to close a file.

E.g. **myfile.close()** here myfile is object of file.

Cont..

read() method

- The read() method reads a string from the file. It can read the data in the text as well as binary format.
- **Syntax : fileobj.read(<count>)**
- Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

```
fileptr = open("file.txt","r");
content = fileptr.read(9);
print(type(content))   #print type of data in file
print(content)
fileptr.close()
```

- Read data up to **count-1**

Cont..

readline() method

- read the file line by line by using a function readline(). The readline() method reads the lines of the file from the beginning,
- i.e., if we use the readline() method two times, then we can get the first two lines of the file.

```
fileptr = open("file.txt","r");
content = fileptr.readline();
print(type(content))
print(content)
fileptr.close()

#read file using loop also
fileptr = open("file.txt","r");
for i in fileptr:
    print(i) # i contains each line of the file
```

Cont..

write() method // for writing data in file

- To write some text to a file, we need to open the file using the open() method with one of the following access modes.
- a: It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.
- w: It will overwrite the file if any file exists. The file pointer is at the beginning of the file. It creates a new file if no file exists.

Syntax : <fileobject>.write(<data>)

e.g. fileptr.write("my first program")

**** you can also take input from the user and write those data into file using write() method**

cont..

```
fileptr=open("myfile.txt","w")

#if will check file pointer is empty or not if not than inside code will executed

if fileptr:
    print("file is open in write mode")

#fileptr.write("my first program using file") #write method is used to write content in file
data1=input("enter some data to write inside text file : ")
fileptr.write(data1)
fileptr=open("myfile.txt","r") # here file open in read mode

if fileptr:
    dataoffile=fileptr.read() #read data from file bit by bit and store into dataoffile object/variable
print (dataoffile) #finally print data of file
```

Cont..

- You can also create new file with “x” mode. It will create a new file using open().
- If file is exist with the same name than it will give an error.

E.g. fileptr=open(“myfile.txt” ,”x”)

File Pointer positions

- tell() method is used to print the byte number at which the file pointer exists.

```
fileptr = open("file2.txt","r")
#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())
content = fileptr.read();
#after the read operation file pointer modifies. tell() returns the location of the
fileptr.
print("After reading, the filepointer is at:",fileptr.tell())
```

Cont..

Reset/modify the file pointer position

- **seek()** is used to modify the file pointer position.

  Syntax : <file-ptr>.seek(offset[, from)

- **offset**: It refers as the new position.
- **from**: It indicates the reference position from where the bytes are to be moved. 0 mean beginning of file, 1 mean from current position, 2 mean from end of file.

```
fileptr = open("file2.txt","r")
#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())
#changing the file pointer location to 10.
fileptr.seek(10);
#tell() returns the location of the fileptr.
print("After reading, the filepointer is at:",fileptr.tell())
```

Cont..

Python os module : is used for renaming, deleting the file and handling the directory.

- rename() method which is used to rename the specified file to a new name.

Syntax : rename(?current-name?, ?new-name?)

e.g. **import os;**

#rename file2.txt to file3.txt

os.rename("file2.txt","file3.txt")

- remove() method which is used to remove the specified file.

Syntax: remove(?file-name?)

E.g.

```
import os;
#deleting the file named file3.txt
os.remove("file3.txt")
```

Cont..

- **Create new directory** : mkdir() method is used to create the directories in the current working directory.

  Syntax : mkdir(?directory name?)

  ```
  e.g. import os;
  os.mkdir("pythonprg")
  ```

- **Changing the current working directory :** The chdir() method is used to change the current working directory to a specified directory.

  Syntax : chdir("new-directory")

  ```
  import os;
  os.chdir("new_dir")
  ```

- **getcwd() method :** returns the current working directory.

  ```
  import os;

  print(os.getcwd())
  ```

Cont..

Deleting directory : rmdir() method is used to delete the specified directory.

Syntax : os.rmdir(?dir_name?)

Eg. Import os;

os.rmdir("python_prg")

File open using with statement :

- introduced in python 2.5. The with statement is useful in the case of manipulating the files.
- is used in the scenario where a pair of statements is to be executed.
- The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

Cont..

- It is always suggestible to use the with statement in the case of file s because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file. It doesn't let the file to be corrupted.
- **Syntax :**

```
with open(<file name>, <access mode>) as <file-pointer>:
```

E.g.

```
with open("file.txt",'r') as f:
    content = f.read();
    print(content)
```

cont..

Write python script/ program output to file

- **check_call()** method of module **subprocess** is used to execute a python script and write the output of that script to a file.

Python_output.py

```
temperatures=[10,-20,-289,100]
def c_to_f(c):
    if c< -273.15:
        return "That temperature doesn't make sense!"
    else:
        f=c*9/5+32
        return f
for t in temperatures:
    print(c_to_f(t))
```

Cont..

Python_store.py

```
import subprocess

with open("output.txt", "wb") as f:
    subprocess.check_call(["python", “python_output.py"], stdout=f)
```

For more about file :

https://www.javatpoint.com/python-files-io

https://www.javatpoint.com/python-read-csv-file

https://www.javatpoint.com/python-write-csv-file

https://www.javatpoint.com/python-read-excel-file

https://www.javatpoint.com/python-write-excel-file