FILE HANDLING

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.

OR

File that stores data in an application

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. (.txt)

- **A text file is simply a sequence of ASCII or Unicode characters.**
- **EOL (new line character i.e. enter) or internal translation occurs**
- **e.g. Python programs, contents written in text editors**

Binary Files-(.dat)

- **A binary file stores the data in the same way as as stored in the memory.**
- **No EOL or internal translation occurs(not converted into other form becoz it is converted into computer understandable form i.e. in binary format)**
- **Best way to store program information.**

. e.g. exe files,mp3 file, image files, word documents

we can't read a binary file using a text editor.

File Handling

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 and .rtf are used as extensions to text files.	Can have any application defined extension.

TEXT FILE

- Text files don’t have any specific encoding and it can be opened in normal text editor itself.
- **Example:**
- **Web standards:** html, XML, CSS, JSON etc.
- **Source code:** c, app, js, py, java etc.
- **Documents:** txt, RTF etc.
- **Tabular data:** csv, tsv etc.
- **Configuration:** ini, cfg, reg etc.

BINARY FILE

- Most of the files that we see in our computer system are called binary files.

- Example:
- Document files: .pdf, .doc, .xls etc.
- Image files: .png, .jpg, .gif, .bmp etc.
- Video files: .mp4, .3gp, .mkv, .avi etc.
- Audio files: .mp3, .wav, .mka, .aac etc.
- Database files: .mdb, .accde, .frm, .sqlite etc.
- Archive files: .zip, .rar, .iso, .7z etc.

- Executable files: .exe, .dll, .class etc.
- All binary files follow a specific format. We can open some binary files in the normal text editor but we can't read the content present inside the file. That's because all the binary files will be encoded in the binary format, which can be understood only by a computer or machine. For handling such binary files we need a specific type of software to open it.
- For Example, We need Microsoft word software to open .doc binary files. Likewise, you need a pdf reader software to open .pdf binary files and you need a photo editor software to read the image files and so on.

File Handling

File operations- adding, modifying, deleting, reading, writing, appending data
There are three steps to perform these operations:

- Open the file.- **Before any reading or writing operation of any file , it must be opened first of all.**

- Process file- i.e perform read or write operation.

- Close the file.-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.

File Handling

open () - built in function

Syntax

file_object/file_handler = open(<file_name>, <access_mode>,< buffering>)

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

access_mode= It is also called file mode. Determines the what kind of operations can be performed with file,like read,write etc.

If no mode is specified then the file will open in read mode.

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

Opening file

```
F=open("notes.txt","r")  #open a file in read mode and specified relative path
F1=open(r"c:\users\hp\notes.txt","r")
#open a file in read mode and specified absolute path(if file is stored in some other folder/location
F1=open("c:\\users\\hp\\notes.txt","w")
```

To specify absolute path of the file either use \\ in each subpath or use r before the path , then python environment will consider it as a raw path string nothing else

```
F1=open(r "c:\users\hp\notes.txt","r")
```

This 'r' has no relation with file mode

File Handling

File opening modes-

Sr. No	Mode & Description
1	**r** - reading only.Sets file 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 . If no file exist, creates a new file for R &W.
8	**wb+** - same as w+ mode but with binary file.
9	**a** -for appending. Move file pointer **at end of the 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. Move file pointer at end. If the file does not exist, it creates a new file for reading and writing.
12	**ab+** - same as a+ mode but with binary mode.

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.

```
f=open("notes.txt","r")
r=f.read()
print(r)
```

Read characters from last position

read([size]) method

It reads the no of bytes

```
f=open(“notes.txt”,”r”)
r=f.read(10)
print(r)
r1=f.read(15)
print(r1)
```

File Handling

Path-

- it is a sequence which gives us access to a file.
- It is an address or location

Absolute Path

The absolute path is the full path to some place on your computer.

OR

It is the path mentioned from the top level of hierarchy.

OR

To access a given file or directory, starting from the root of the file system

For example: Absolute path: C:\Users\hp\Desktop\cs\function.py

Relative Path

The relative path is the path to some file with respect to current working directory

e.g. Relative path: "function.py"
or
"..\function.py"

The readline() Method

It reads the lines.**It returns the read lines as list**

```
f=open("notes.txt","r")
r=f.readline()
print(r)
```

Combine Open() and read ()

```
f=open(“notes.txt”,”r”)
r=f.read()
print(r)

f=open(“notes.txt”,”r”).read()
print(f)
f=open(“notes.txt”,”r”).readline()
print(f)
f=open(“notes.txt”,”r”).readlines()
print(f)
```

Read first 2 lines

It reads the lines.It returns the **read lines as list**

```
f=open(“notes.txt”,”r”)
r=f.readline()
print(r)
r1=f.readline()
print(r1)
```

OR

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

```
f=open(“notes.txt”,”r”)
r=f.readlines(2)
print(r)
```

File Handling

The close() Method

close(): Used to close an open file..

```
f=open(“notes.txt”,”r”)
r=f.read()
print(r)
f.close()
```

- close() breaks the link of file object
- After using this method, an opened file will be closed and a closed file cannot be read or written any more.
- Closing file is important
- Files are automatically closed at the end of the program
- But it is good to close file explicitly.
- This can boost performance.

File Handling

Program to display number of lines in a file.

```
f=open("c:\\users\\hp\\notes.txt","r")
r=f.readlines()
d=len(r)
print(d)
f.close()
```

File Handling

write() and read() based program

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

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

OUTPUT

Welcome to python
Regularly visit pythonapsdk.blogspot.com

File Handling

Iterating over lines in a file
e.g.program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python'
 f.write(line1)
line2="pythonapsdk.blogspot.com"
f.write(line2)
f.close()

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

File Handling

Processing Every Word in a File

e.g.program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python'
f.write(line1)
line2="pythonapsdk.blogspot.com"
f.write(lin
e2)
f.close()

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

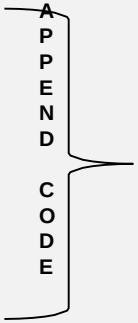
File Handling

Append content to a File

```
f = open("a.txt", 'w')
line = 'Welcome to pythonapsdk.blogspot.com’
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()
```



File Handling

FLUSH()

It forces the writing of data on disc still pending in buffer

```
f = open("a.txt", 'w')
line = 'Welcome to pythonapsdk.blogspot.com’
f.flush()
D=“class xii”
f.write(D)
f.write(“section L”)
f.flush()
f.close()
```

File Handling

File Pointer

it tells the current position in the file where writing or reading will take place.(like a bookmark in a book)

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.

```
f = open("a.txt", 'w')
line = 'Welcome to pythonapsdk.blogspot.com'
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()
```

File Handling

Standard input, output, and error streams in python

Input-keyboard-standard input device(stdin)

Output-monitor-standard output device(stdout)

Error-monitor-standard error device(stderr)

Physically devices implemented internally as stdin,stdout,stderr files in python.

e.g.program

```
import sys
f=open("notes.txt","r")
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

With statement- no need to call close()

```
With open("notes.txt","w") as f:
        f.write("hello world")
```

File Handling

CSV file-comma separated values

```
import csv
with open('studata.csv','a') as cs:
        r=csv.writer(cs)
        for i in range(2):
                n=input('enter name:')
                m=input('enter address:')
                r.writerow([n,m])
```

File Handling

Methods of os module

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())
```