Python Notes

_/_/___

* File Management in Python :-

- Python too suports file handling and allows users to handle files i.e. to read and write files, along with many other file handling options, to operate on files.

- file handling Plays an important role when the data needs to be stored permanenetly into the file.

Operations on files :- firstly, we need to open the file first for reading or writing data into it. After performing operations on the file or saving data into it, it needs to be closed.

- Hence, in Python, a file operation takes place in the following order :-

1. Open a file

2. Read or Write (Perform operation)

3. Close a file.

__/__/____

1 Opening a File :-

- In Python, a file can be opened by open() function.

- It is a very simple built-in function of Python to open a file.

Syntax:-
file object = open(filename, access_mode, buffering)

- file_name :- It contains the name of the file that the programmer wishes to access.

- access_mode :- It 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.

Example of Opening a file using Python open()

```
>>> fp = open("data.txt")
>>> fp = open("C:/PythonPrograms/data.txt")
```

__/__/____

* File Modes :-

At the time of opening, we have to specify the mode, which represents the purpose of the opening file.

f = open (filename, mode)

Some of the modes used in Python :-

1. r :- Opening a file for reading only. Open an existing file, This is the default mode.

2. w :- Open an existing file for a write operation.

3. a :- Open an existing file for append operation.

4. r+ :- Opens a file for both reading and writing

5. w+ :- To write and read data. It will override existing data. If the file does not exist, creates a new file reading / writing.

6. a+ :- To append and read data from the file.

Example :-

_/_/_

a file named "jp", will be opened with reading mode

```
file = open ("jp-txt", 'r')

for each in file:
        print (each)
```

* File Object Attributes

A file object has a lot of attributes. After opening a file, the programmer can access information about the file using various available attributes.

Attribute
(fp is the file object)

fp.closed :- Returns true if the file is closed, false otherwise.

fp.mode :- Returns access mode with which file was opened.

fp.name :- Returns name of the file.

fp.softspace :- Returns false if space explicity required with print, true otherwise.

Example

```
fp = open ("jp-txt", "wb")   ← writing only in binary form
print ('Name of the file:', fp.name)
print (' Closed or not!', fp.closed)
print ('Opening mode:', fp.mode)
```

_/_/_

* File Encoding

- Encoding we mean either text mode or binary mode.

- When working with files in text mode, it is recommended to specify the encoding type.

- Files that are stored in bytes on the disk, we need to decode them into str (text) when we read into Python.

- Encoding is also performed while writing texts to the file.

Encoding scheme for windows :- 'cp1252'
" " " LINUX :- 'utf-8'

Example of file encoding :-

```
fp = open ("jp.txt", mode='r', encoding='cp1252')
                                   for↑ windows
fp= open ("jp.txt", mode='r', encoding='utf-8')
                           # for LINUX
```

__/__/____

* Closing a File

- The file can be closed by using the close() function.
- It is a built in function provided by Python for manually closing a file.
- It releasing all the resources aquired by it.

Syntax

```
fileobject.close()
```

Example

```
file = open ("sample.txt")
print (file.read())
file.close()
file.write ("Attempt to write on a closed file")
```

Output

```
Value Error: I/O Operation on closed file.
```

__/__/__

* read() & write() Methods :-

For writing data into file write() method is used. For reading data from the file read() method is used.

• Writing to a file :-

Writing data to the file is accomplished by using write() method. This method returns the number of characters written to the file.

Syntax : fileobject.write(string)

Program

keyword

```
with open("jp.txt" ,'w', encoding='CP1252') as fp:

fp.write ("Notes by jpwebdevelopers \n")
fp.write (" Example of writing \n")
fp.close()
```

• Reading from a file :-

Read() method is used to read the contents of a file. But, before that the file must be opened in read mode 'r'.

__/__/____

Syntax:- fileobject.read(size)

There are three functions of reading data from a file represented by file object f.

```
data = f.read()        # reads entire file contents
data = f.read(n)       # reads n characters
data = f.readline()    # reads a line.
```

$\overset{\text{size}}{n}$

\boxed{\text{Program}}

```
with open ("jp.txt", 'r', encoding = 'cp1252') as fp:
Print (fp.read(5))
print(fp.read(5))
fp. close()
```

Example of \boxed{\text{readline()}} method

```
with open ("jp.txt", 'r', encoding = 'cp1252 ') as fp:
print (fp.readline())
fp. close()
```

_/_/___

* tell() and seek() Methods

tell() This method is used to determine the current position of a file pointer. It can be used either while writing to a file or reading from the file.

Syntax fileobject.tell()

seek() This method is used to move the file pointer to a particular location in the file.

Syntax Seek(offset [, from])

offset: represents the number of bytes to be moved from the specified Position.

from	value
Beginning	0
Current	1
End	2

Program of tell() and seek()

```
with open("jp.txt", 'r', encoding='cp1252')
                                   as fp:
    Print (fp.read(4))
    pos= fp.tell()
    print ('Current Position:', pos)
    fp.seek (0,1)
    print(fp.readline())
    fp.close()
```

* Renaming and Deleting files :-

Python language provides special methods for renaming and deleting a file. These methods are available through the Python os module. For using these methods, the Programmer needs to import the os module.

(i) rename() method This method takes two arguments.
Syntax :- os.rename(oldfilename, newfilename)

Example

```
import os
os.rename('data.txt', 'info.txt')
```

(ii) remove() method This method is used to delete the file. It has only one argument, which is the file name to be deleted.

Syntax os.remove(filename)

Example of remove method

```
import os
os.remove('info.txt')
```

* Directories in Python :=

We can also work on directories in Python. This is acheived by making the use of os module, and various methods in it.

Here we discuss some methods associated with the directories in Python

(i) mkdir() method

- This mkdir() method is used to create directories in the current directory.

- This method contains only one argument that is the name of the directory to be created.

Syntax :- os.mkdir ("new_directory_name")

# Example of mkdir() method

```
import os
os.mkdir('datafiles')
```

_/_/_

(ii) chdir() method

- This method is used to change the current directory.
- It also take one argument for moving one directory to another and making it the current directory.

Syntax:- os.chdir('newdirectory')

Example

```
import os

os.mkdir('datafiles/newfiles')
os.chdir('datafiles/newfiles')
fp = open('input.txt', 'w')
fp.write('Hello', JPWebdevelopers')
fp.close()
```

(iii) getcwd() method

The getcwd() method is used to obtain the information of the presently working directory.

__/__/____

Syntax os.getcwd()

Program of getcwd() method

```
import os

Print(os.getcwd())
os.chdir('datafiles')
Print(os.getcwd())
os.chdir('newfiles')
Print(os.getcwd())
```

(IV) rmdir() method

The rmdir() method is used to remove or delete the directory, which is mentioned in its argument.

Syntax:- os.rmdir('directoryname')

Illustration of rmdir() method

```
import os

os.chdir('datafiles')
os.rmdir('newfiles')
```

__/__/____

(V) listdir() method

The listdir() method is used to get the list of all files and directories in the currently working directory.

Syntax os.listdir()

Illustration of listdir() method

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