FILE HANDLING

- **What is a file?**

A file is some information or data which stays in the computer storage devices. ... Python gives you easy ways to manipulate these files. Generally we divide files in two categories, text file and binary file. Text files are simple text where as the binary files contain binary data which is only readable by computer.
Hence, in Python, a file operation takes place in the following order.
Open a file
Read or write (perform operation)
Close the file

- **Opening a file**

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.
syntax : open(filename, mode).
There are three kinds of mode, that Python provides and how files can be opened:
" r ", for reading.
" w ", for writing.
" a ", for appending.
" r+ ", for both reading and writing

example:

```
>>> f = open("test.txt")     # open file in current directory
>>> f = open("C:/Python33/README.txt")  # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode.
The default is reading in text mode. In this mode, we get strings when reading from the file.
On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

- **Reading data from a file**

There is more than one way to read a file in Python. If you need to extract a string that contains all characters in the file then we can use file.read(). The full code would work like this:

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

- **Writing data to a file**

Example:1

```
file = open('myfile.txt','w')
file.write("This is the write command")
file.write("It allows us to write in a particular file")
```

Example2:

```
def fun(path,text):
    with open(path,'w') as f:
        f.write(text)
        read(path)
def read(path):
    with open(path, 'r') as f2:
        data= f2.read()
        print(data)

fun("D:/naveen/programs/myfile.txt",'hello naveen')
read("D:/naveen/programs/myfile.txt")
```

- **Closing a file**

When we are done with operations to the file, we need to properly close the file. Closing a file will free up the resources that were tied with the file and is done using Python close() method.

Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

Example:

```
file = open('myfile.txt','w')
file.write("This is the write command")
file.close()
```

- **Working with the methods of file obJect**

A file object allows us to use, access and manipulate all the user accessible files. One can read and write any such files.

open(): Opens a file in given access mode.

```
open(file_address, access_mode)
```

Examples of accessing a file: A file can be opened with a built-in function called open(). This function takes in the file's address and the access_mode and returns a file object.

There are different types of access_modes:

r : Opens a file for reading only
r+ : Opens a file for both reading and writing
w : Opens a file for writing only
w+: Open a file for writing and reading.
a : Opens a file for appending
a+ : Opens a file for both appending and reading

When you add 'b' to the access modes you can read the file in binary format rather than the default text format. It is used when the file to be accessed is not in text.
read([size]): It reads the entire file and returns it contents in the form of a string. Reads at most size bytes from the file (less if the read hits EOF before obtaining size bytes). If the size argument is negative or omitted, read all data until EOF is reached.
Example:

```
with open("C:/Users/sande/Desktop/Amazon.txt", 'r') as f:
    text = f.read(123)
    print(text)

f.close()
```

output:

```
ï»¿2/11/2020

Amazon.in - Order 403-4664037-2109106

Details for Order #403-4664037-2109106
Print this page for your record
```

readline([size]): It reads the first line of the file i.e till a newline character or an EOF in case of a file having a single line and returns a string. If the size argument is present and non-negative, it is a maximum byte count (including the trailing newline) and an incomplete line may be returned. An empty string is returned only when EOF is encountered immediately.

Example:

```
with open("C:/Users/sande/Desktop/Amazon.txt", 'r') as f:
    text = f.readline()
    print(text)
```

```
f.close()
```

output:

ï»¿2/11/2020

seek(offset, from_where): It is used to change the file object's position. Offset indicates the number of bytes to be moved. from_where indicates from where the bytes are to be moved.

Example:

```
file=open("C:/Users/sande/Desktop/mydata.txt")
file.seek(17,0)
a=file.read(7)
print(a)
file.close()
```

- **Replacing the content of file**

1. Open input file in read mode and handle it in text mode.
2. Open output file in write mode and handle it in text mode.
3. For each line read from input file, replace the string and write to output file.
4. Close both input and output files.

Example:

```
fin = open("C:/Users/sande/Desktop/data.txt", "rt")
fout = open("C:/Users/sande/Desktop/out.txt", "wt")

for line in fin:
        fout.write(line.replace('pyton', 'python3'))

fin.close()
fout.close()
```

- **Working with Directories**

If there are a large number of files to handle in your Python program, you can arrange your code within different directories to make things more manageable. A directory or folder is a collection of files and sub directories. Python has the os module, which provides us with many useful methods to work with directories (and files as well).

Get Current Directory

We can get the present working directory using the **getcwd()** method.

This method returns the current working directory in the form of a string

Example:

```
>>> import os
>>> os.getcwd()
'D:\\naveen\\programs'
```

Changing Directory

We can change the current working directory using the chdir() method.

The new path that we want to change to must be supplied as a string to this method. We can use both forward slash (/) or the backward slash (\) to separate path elements.

Example:

```
>>> os.chdir('C:\\Python33')
>>> print(os.getcwd())
C:\Python33
```

List Directories and Files

All files and sub directories inside a directory can be known using the **listdir()** method.

This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

Example:

```
>>> print(os.getcwd())
C:\Python33

>>> os.listdir()
['DLLs',
'Doc',
'include',
'Lib',
'libs',
'LICENSE.txt',
'NEWS.txt',
'python.exe',
'pythonw.exe',
'README.txt',
'Scripts',
```

'tcl',
'Tools']

Making a New Directory

We can make a new directory using the mkdir() method.
This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.
example:

```
>>> os.mkdir('test')
>>> os.listdir()
['test']
```

Renaming a Directory or a File

The rename() method can rename a directory or a file.
The first argument is the old name and the new name must be supplies as the second argument.

```
>>> os.listdir()
['test']
>>> os.rename('test','new_one')
>>> os.listdir()
['new_one']
```

Removing Directory or File

A file can be removed (deleted) using the remove() method.
Similarly, the rmdir() method removes an empty directory.

```
>>> os.listdir()
['new_one', 'old.txt']
>>> os.remove('old.txt')
>>> os.listdir()
['new_one']
>>> os.rmdir('new_one')
>>> os.listdir()
[]
```

However, note that rmdir() method can only remove empty directories.
In order to remove a non-empty directory we can use the rmtree() method inside the shutil module.

```
>>> os.listdir()
```

['test']

>>> os.rmdir('test')
Traceback (most recent call last):
...
OSError: [WinError 145] The directory is not empty: 'test'

>>> import shutil
>>> shutil.rmtree('test')
>>> os.listdir()
[]

- **Handling IO Exceptions**

IOError

Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.

Raised for operating system-related errors.

```
try:
    f = open("fake.txt", mode="r")
except IOError:
    print("IOError")
```

output:

IOError