File Handling in Python

Introduction

- A file is a collection of data stored on secondary storage device like hard disk.
- So far we had been processing data that was entered through the keyboard using input()
- Reading from input is become very tedious especially when there are large amount of data to be processed.
- To overcome this to find a better solution is to combine all input data into a file and then read this data from file whenever we required.
- When a program is being executed , its data is stored in '**random access memory** ' (RAM) ,it can accessed faster by CPU, it is also **volatile**.
- **Volatile** ,which means that when the program ends, or the computer shuts down, all the data is lost.
- To rectify this we need to store data in **non-volatile** or **permanent** storage media (Hard disk,USB,DVD .etc)

What is a file?

- **A file is a collection of data** on secondary storage device (non-volatile) is stored in named locations on the device or media called **files**.

  Example : How use notebook? To open - To read or write – To finally close.
- The Notebook concept can be applied to files , that is we first open a file, read or write to it , and then finally close it.
- Hence, in Python, a file operation takes place in the following order.

  1. **Open a file** 2. **Read** or **Write** (perform operation) 3. **Close** the file
- A file is basically used real life applications involve huge amount of data and in such situations the console oriented I/O operations pose two major problems.
  - ✓ 1. It becomes cumbersome and time consuming to handle huge amount of data through terminal
  - ✓ 2. Using I/O operations with terminal ,the entire data is lost when either program or computer is terminated. It becomes necessary to store data on permanent storage and read whenever necessary with out destroying the data.
- How data is read or written to a file I/O operations same as terminal also

Types of files(**text** & binary)

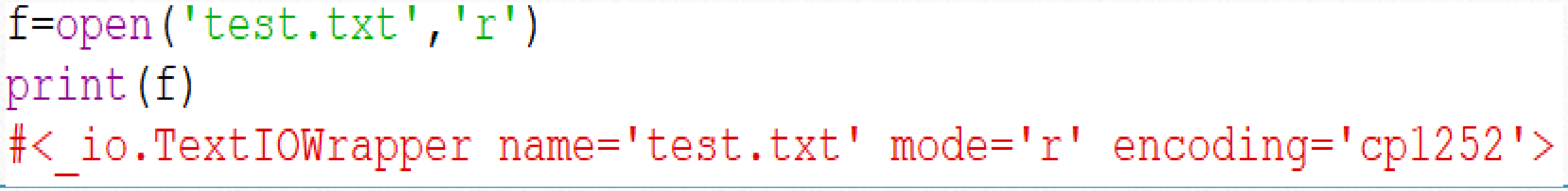
- Python as supports two type of files → **text** files and **binary** files
- A **tex**t file is a stream of characters that can be sequentially processed by a computer in forward direction.
- A text file can process characters they can only read or write data one character at a time, the newline characters may be converted to or from carriage- return /linefeed.
- In a text file, each line of data end with a newline character .Each file ends with a special character called the end - of - file (EOF) marker.

Binary File

- In this type of file, there is no terminator for a line and the data is stored after converting it into machine understandable binary language
- A binary file is a file which may contain any type of data . Encoded in binary form for computer storage and processing purposes.
- It includes file such as word processing documents, PDFs, images, spreadsheets, videos, zip files, and other executable programs.
- Text file contains only basic characters ,do not store any information about color, font and size of text whereas binary files are mainly used to store data beyond such as images, executables etc.

Opening a file

- Before reading or writing a file , we must first open it using python's built-in **open()** function, this function creates file object, which will be used to invoke methods associated with it.
- The syntax : - fileObj = **open(file_name**,[**access_mod**] **)**
- Here **file_name** is a string value that specifies name of the file that you want to access
- **access_mode** indicates the mode in which the file has to be opened i.e., read, write, append, etc. The **open()** function returns a file object. This file object will be used to read, write, or perform any other operation on the file. It works like a file handle.
- Access_mode is an optional parameter and the default file access mode is read(r).



```
f=open('test.txt','r')
print(f)
#<_io.TextIOWrapper name='test.txt' mode='r' encoding='cp1252'>
```

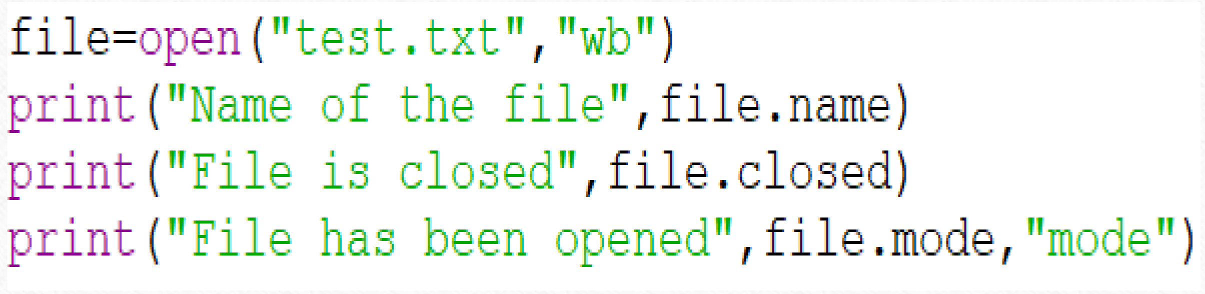
Access Modes

Mode	Purpose
r/rb	This is default mode of opening a file which opens the file for reading only ,pointer is place at be beginning
r+/rb+	This mode opens a file for both reading and writing in binary format. Pointer is placed at beginning of the file(in binary format also)
w/wb	This mode opens the file for writing only. when a file is opened in **w** mode. Two things can happen.if the file doesn't exists , a new file is created for writing, if the file already exists and has some data stored in it.that contains overwritten. (in binary also)
w+/wb+	Opens a file both reading and writing .when a file is opened in this mode, two things can happen. If the file doesn't , a new file is created for reading as well as writing, if the file already exists and has some data stored in it.that contains overwritten(in binary format also)
a/ab	Opens a file for appending .The file pointer is place at the end of the file if the file exists , If the file does not exist it create a new file for writing(in binary format also)
a+/ab+	Opens a file in reading and appending .The file pointer is place at the end of the file if the file exists , If the file does not exist it create a new file for reading and writing(in binary format also)

The File Object Attributes

- Once a file is successfully opened , a file object is returned . Using this object, you can easily access different types of information related to that file.
- This information can be obtained by reading values of specific attributes of the file

Attribute	Information Obtained
fileObj.closed	Returns True if the file is closed and False otherwise
fileObj.mode	Returns access mode with which file has been open
fileObj.name	Return name of the file



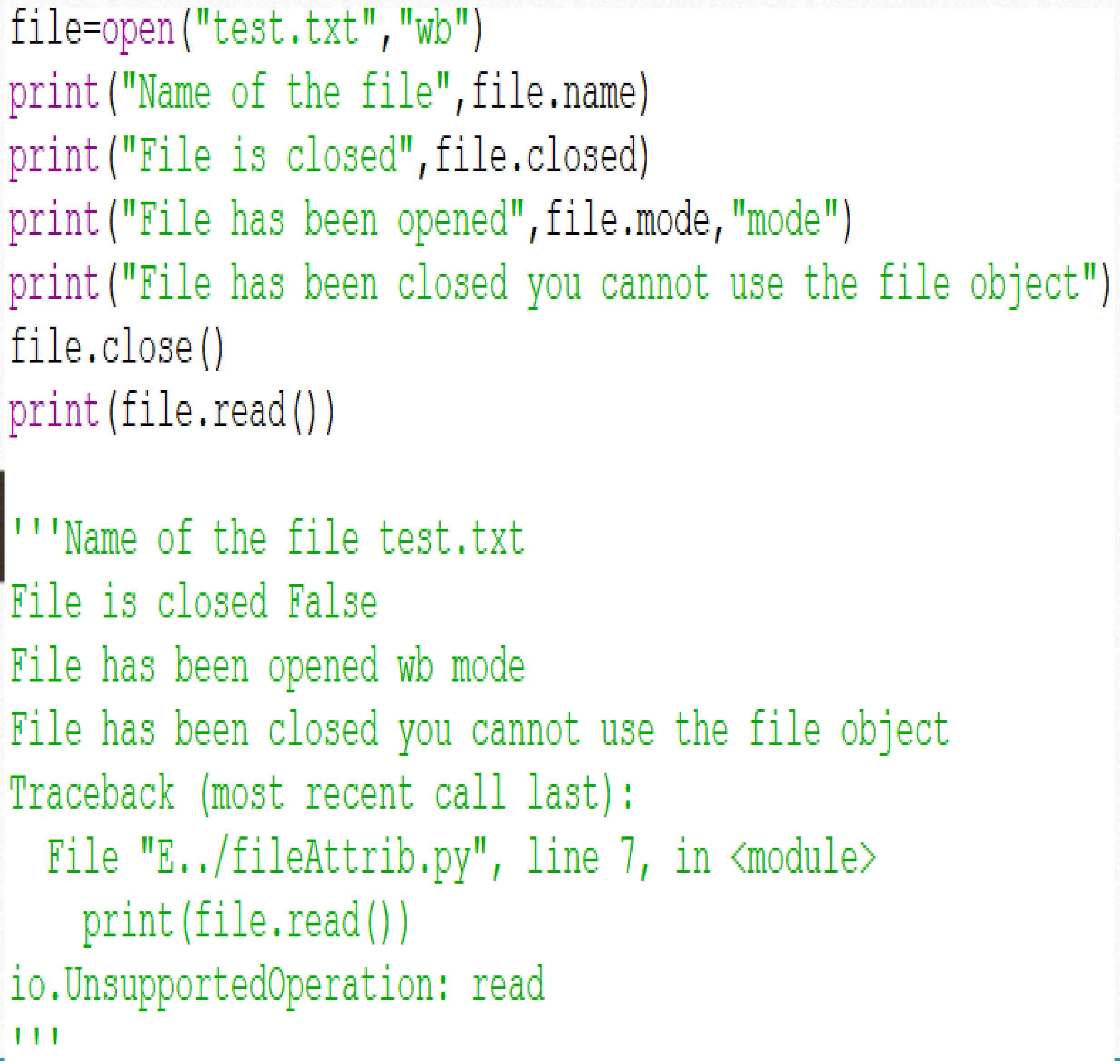
```
file=open("test.txt","wb")
print("Name of the file",file.name)
print("File is closed",file.closed)
print("File has been opened",file.mode,"mode")
```

The close() Method

- The close() method as the name suggests is used to closed the file object. Once a file object is closed , you cannot further read from or write into the file associated with the file object.
- While closing the file object the close() flushes any unwritten information.
- Python automatically closes a file when the reference object is of a file is reassigned to another file, but as a good programming habit you should always explicitly use the **close()** method to close a file
- Syntax:

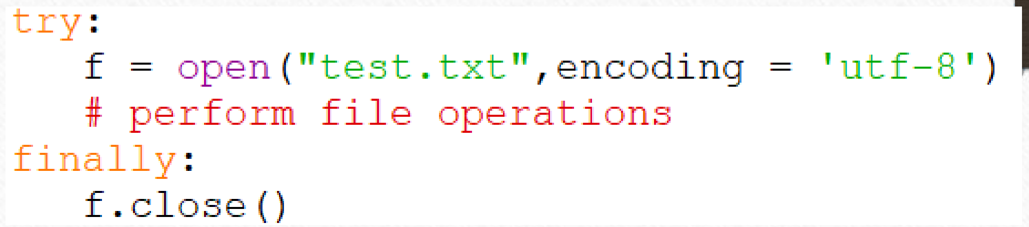
```
fileObj.close()
```

- Once the file is closed using the **close()** method ,any attempt to use the file object will result in an error.



```
file=open("test.txt","wb")
print("Name of the file",file.name)
print("File is closed",file.closed)
print("File has been opened",file.mode,"mode")
print("File has been closed you cannot use the file object")
file.close()
print(file.read())

'''Name of the file test.txt
File is closed False
File has been opened wb mode
File has been closed you cannot use the file object
Traceback (most recent call last):
  File "E../fileAttrib.py", line 7, in <module>
    print(file.read())
io.UnsupportedOperation: read
'''
```

- This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.
- A safer way is to use a **try...finally** block.



```
try:
   f = open("test.txt",encoding = 'utf-8')
   # perform file operations
finally:
   f.close()
```

- This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop.

Reading and Writing Files

- The read() and write() are used to read data from file and write data to files respectively.

write()

- The write() method is used to write a string to an already opened file. Of course this string may include numbers, special characters, or other symbols.
- While writing data to a file , you must remember that the **write()** method does not add a newline**('\n')** to the end of the string .

write() : To write a single string to a text file

Syntax: `fileObj.write(str1)`

writelines() : It takes a list of strings and write them to a file. Used to insert multiple strings at a single time.

Syntax: `fileObj.writelines(L)`

- Example -1 (write() method)

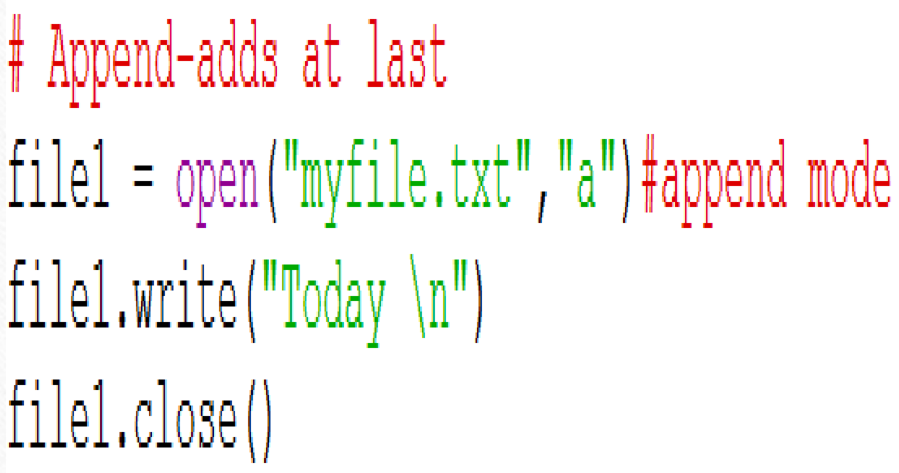
```
file=open("File1.txt","w")
file.write("Welcome to File Handling\n")
file.write("Hello all, hope you are enjoying")
file.close()
print "data written...."
```

- Example -2 (writelines() method)

```
file=open("File1.txt","w")
lines = ["Welcome to File Handling","Hello all, hope you are enjoying"]
file.writelines(lines)
file.close()
print "data written...."
```

append() Method

- Once you have stored some data in a file , you can always open that file again to write more data or append data to it.
- To append a file , you must open it using 'a' or 'ab' mode depending on whether it is a text file of binary file. 'open with **w'** or **'wb'** mode then start writing data into it . Then its existing contents would be overwritten.
- Open the file in **'a'** or **'ab'** mode to add more data to existing data stored in a file and appending data is especially essential when creating a log of events or combining a large set of data into file

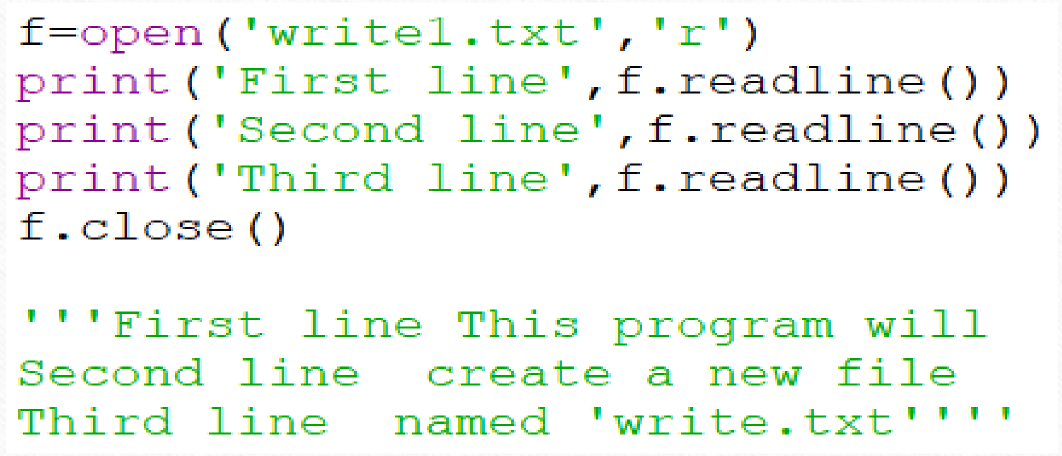


```
# Append-adds at last
file1 = open("myfile.txt","a")#append mode
file1.write("Today \n")
file1.close()
```

Reading from a file

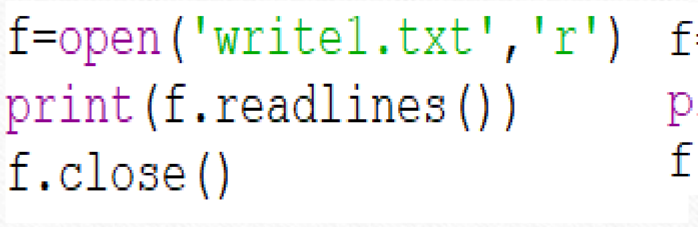
- The *read()* method is used to read a string from an already opened file.
- There are three ways to read data from a text file.
  - **read() :** Returns the read bytes in form of a string. Reads n bytes, if no n specified, reads the entire file.
    - `fileObj.read([n])`
  - **readline() :** Reads a line of the file and returns in form of a string. For specified n, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.
    - `fileObj.readline([n])`
  - **readlines() :** Reads all the lines and return them as each line a string element in a list.
    - `fileObj.readlines()`

- The **readline()** method is used to read a single line from the file, the method returns an empty string when the end of the line has been reached.
- The blank line is represented by \n and the **readline()** method returns a string containing only newline character when a blank line is encountered in the file
- After reading a line from the file using the readline() method, the control automatically passes to the next line.

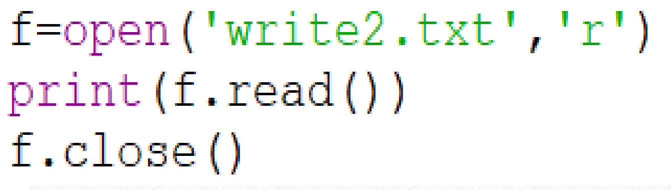


```
f=open('write1.txt','r')
print('First line',f.readline())
print('Second line',f.readline())
print('Third line',f.readline())
f.close()

'''First line This program will
Second line  create a new file
Third line  named 'write.txt'''
```

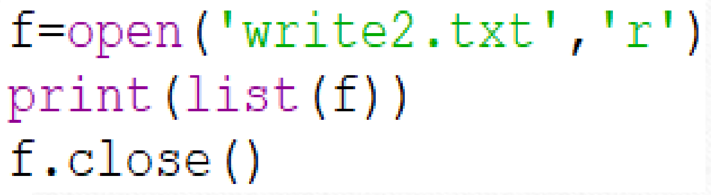
- The **readlines() or read()** method is used to read all lines in the file and display the content of the file using **list()** method.



```
f=open('write1.txt','r')
print(f.readlines())
f.close()
```



```
f=open('write2.txt','r')
print(f.read())
f.close()
```



```
f=open('write2.txt','r')
print(list(f))
f.close()
```

Reading Information From Files using loops

- Each execution of the loop will read a line from the file into a string

Format:

```
for <variable to store a string> in <name of file variable>:
    <Do something with the string read from file>
```

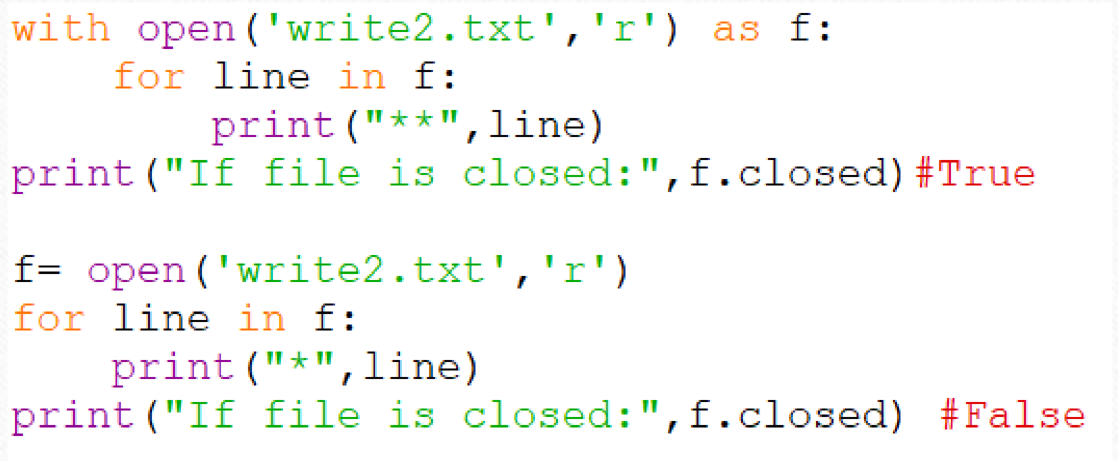
Example:

```
file=open("File.txt","r")
for line in file:
    print(line)  # Echo file contents back onscreen
file.close()
```

Using with keyword

- It is good programming habit to use the **with** keyword when working with file objects
- That the file is properly closed after it is used even if an error occurs during read or write operation or even when you forget to explicitly close the file.
- The file is opened using **with** keyword
- It is automatically closed after for loop is

  over
- The file is opened using **without** keyword
- It is not automatically closed ,we need to explicitly close the file after using it.
- When you open a file for reading or writing, the file is searched in the current working directory, if the file exists somewhere else then you need to specify the path of the file



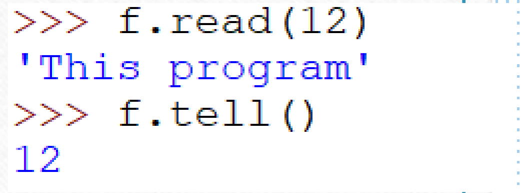
```
with open('write2.txt','r') as f:
    for line in f:
        print("**",line)
print("If file is closed:",f.closed)#True

f= open('write2.txt','r')
for line in f:
    print("*",line)
print("If file is closed:",f.closed) #False
```

File Positions

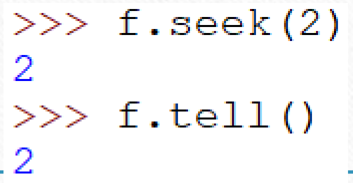
- The file management system is associates a pointer often know as file pointer that facilitates the movement across the file for reading and writing data.
- The **tell()** method tells the current position within the file at which the next read or write operation will occur, it specifies number of bytes from the beginning.
- The **seek(offset[,from])** method is used to set the position of the file pointer of in simpler terms, move the file pointer to a new location.
- The **offset** argument indicates number of bytes to be moved and the from argument specifies the reference position form where the bytes are to be moved.
- **From arguments** 0 – From the beginning of the file

  1 – From the current position of the file

  2 – From the end of the file

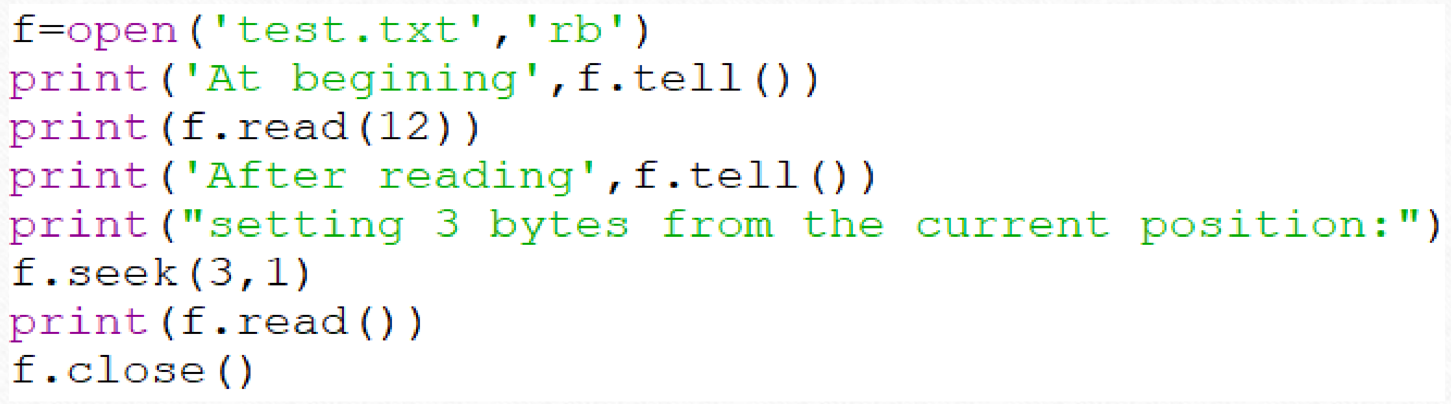


```
>>> f.read(12)
'This program'
>>> f.tell()
12
```



```
>>> f.seek(2)
2
>>> f.tell()
2
```

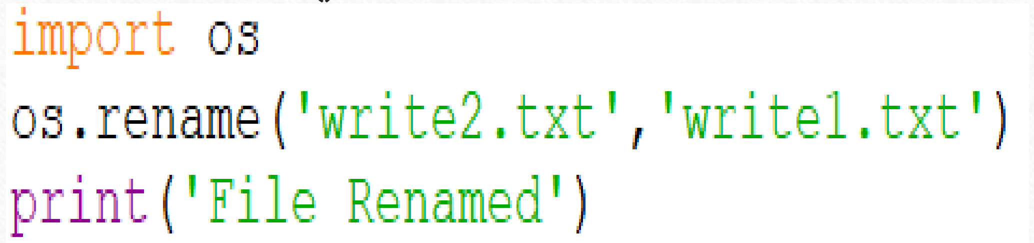
- **File Positons**



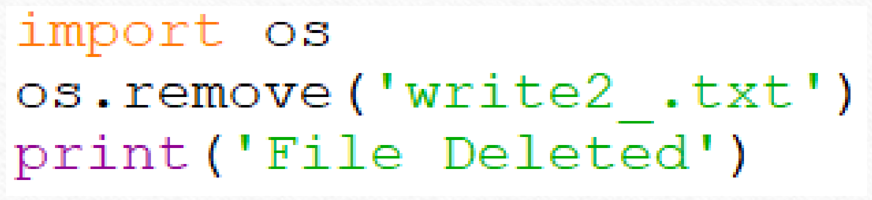
```
f=open('test.txt','rb')
print('At begining',f.tell())
print(f.read(12))
print('After reading',f.tell())
print("setting 3 bytes from the current position:")
f.seek(3,1)
print(f.read())
f.close()
```

Renaming and deleting Files

- To perform file – processing operations like renaming and deleting files, to use methods defined in the os module.
- The **rename()** methods takes two arguments, the current filename and the new filename
- The **remove()** method take a filename as an argument and deletes that file.



```
import os
os.rename('write2.txt','write1.txt')
print('File Renamed')
```



```
import os
os.remove('write2_.txt')
print('File Deleted')
```

Practice Problems

1. Write a program that copies first 10 bytes of a binary file into another?
2. Write a program that copies one python script into another in such a way that all comment lines are skipped and not copied in the destination file.
3. Write a program that accepts filename as an input from the user. Open the file and count the number of times character appears in the file?
4. Write a program that reads data from a file and calculates the percentage of vowels and consonants in the file?