Reading and Writing Files

Python provides many functions for reading and writing the open files.

Read from files

Python provides mainly three types of read functions to read from a data file. But before you can read from a file, the file must be opened and linked via a file object or file handle.

```
    read()        <filehandle>.read([n])
```

this function reads at most n bytes; if no n is specified, reads the entire file.
Return the read bytes in the form of string.

```
    readline()        <filehandle>.readline([n])
```

reads a line of input; if n is specified reads at most n bytes. Return the read bytes in the form of a string **or return a blank string if no more bytes are left for reading in the file**.

```
    readlines()  <filehandle>.readlines()
```

reads all lines and returns then in a **list**.

Reading a file's first 30 bytes and printing it

```
>>> myfile = open(r"e:\poem.txt", "r")

>>> str = myfile.read(30)

>>> print(str)
```

```
THE SONNETS

by William Shakesp
```

if need to perform just a single task with the open file, then you may combine the two steps of opening the file and performing the task.

```
    str = open(r"e:\poem.txt", "r").read(30)
```

would give the same result as that of above code.

Reading n bytes and then reading some more bytes from the last position read

```
    >>> myfile = open(r"e:\poem.txt", "r")
    >>> str = myfile.read(30)
    >>> print(str)
THE SONNETS
by William Shakesp
    >>> str2 = myfile.read(50) # read next 50 bytes
    >>> print(str2)
    eare
                    1
    From fairest creatur
```

Reading a file's entire content

```
    >>> str = open(r"e:\poem.txt", "r").read()

# see when no value is specified as read()'s argument,

# entire file read.

    >>> print(str)

THE SONNETS
by William Shakespeare
1
From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thy self thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
```

And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
Pity the world, or else this glutton be,
To eat the world's due, by the grave and thee.

Reading a file's first three lines - line by line

```
    >>> myfile = open(r"e:\poem.txt")
    >>> str1 = myfile.readline()
    >>> print(str1)
```

```
THE SONNETS
```

```
    >>> str2 = myfile.readline()
    >>> print(str2)
```

```
by William Shakespeare
```

```
>>> str3 = myfile.readline()
>>> print(str3)
```

```
        1
```

Reading the complete file - line by line

```
myfile = open(r"e:\poem.txt")
while (str != ""):
    str = myfile.readline()
    #  readline() method return a blank string if no more
    #bytes are left for reading in the file
    print(str)
myfile.close()
```

Reading the complete in a list

```
myfile = open(r"e:\poem.txt")
s = myfile.readlines()
```

```
# notice it is readlines() has read the entire file in a
#list of strings
print(s)
myfile.close()
```

Display the size of a file in bytes.

```
myfile = open(r"e:\poem.txt")
s = myfile.read()
size = len(s)
print("Size of the given file :", size, "bytes")
myfile.close()
```

```
Size of the given file : 6737 bytes
```

Display the number of lines in the file

```
myfile = open(r"e:\poem.txt")
s = myfile.readlines()
#readlines() return the list of all lines
size = len(s)
print("Number of lines in file  :", size, "lines")
myfile.close()
```

```
Number of lines in file  : 172 lines
```

Writing onto Files

Like reading functions, the writing functions also work on open files, ie the files that are opened and linked via file-object or file-handle.

```
    write()        <filehandle>.write(str1)
```

Writes string to file referenced by <filehandle>

```
    writelines()      <filehandle>.writelines(L)
```

Write all strings in list L as lines to file referenced by <filehandle>.
In Python, writing in file can be take place in the following forms:
i) In an existing file, while retaining its content

a) If the file has been opened in append mode ("a") to retain the old content.

b) If the file has been open in 'r+' or 'a+' modes to facilitate reading as well as writing.

ii) To create a new file or to write on an existing file after truncating/overwriting its old content

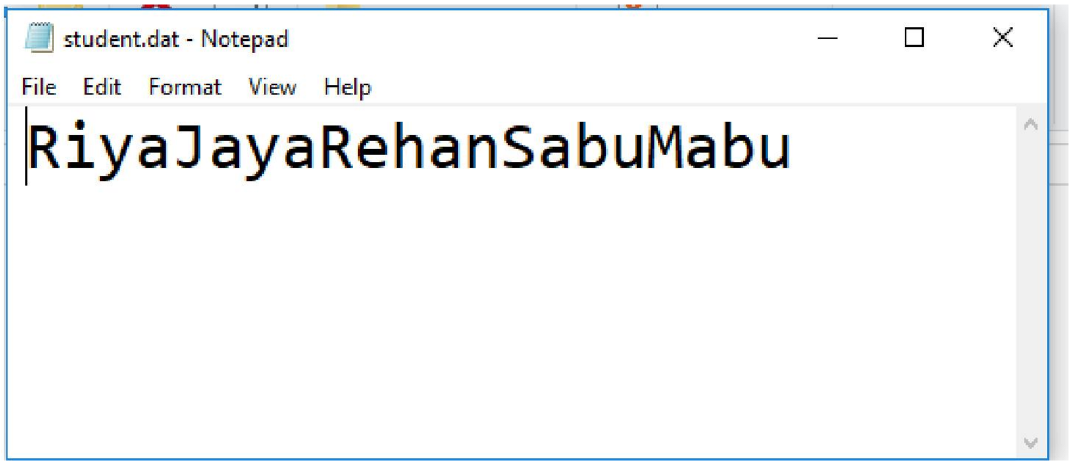
a) If the file has been opened in write-only mode("w").

b) If the file has been open in "w+" mode to facilitate writing as well as reading.

iii) Make sure to use close() function on file-object after you have finished writing. The contents remains in memory buffer and to force-write the content on file.

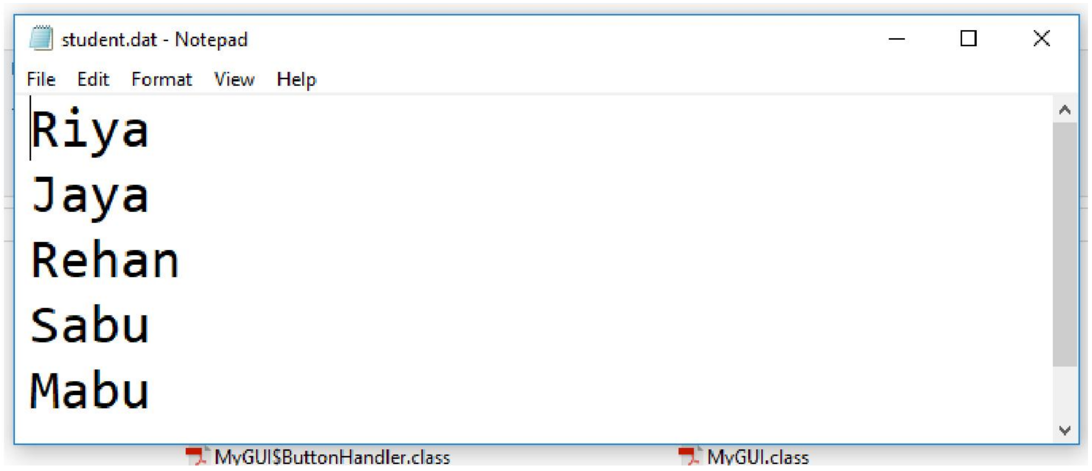
```
fileout = open("student.dat", "w")
for i in range(5):
    name = input("Enter name of student")
    fileout.write(name)
fileout.close()
```

This file created at the location of source, however if you want to create the file in specific folder then you must specify the file-path. We can see the output of file in Notepad, the file-contents that write() does not add any extra character like('\n') after every write operation. Thus to group the contents you are writing in file, line wise, make sure to write new line character on your own.



```
RiyaJayaRehanSabuMabu
```

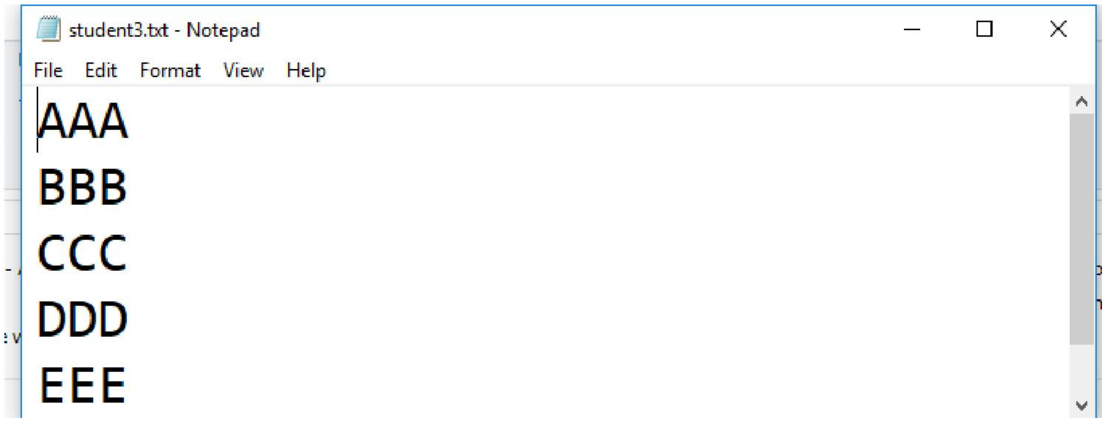
```
fileout = open("student.dat", "w")
for i in range(5):
    name = input("Enter name of student")
    fileout.write(name)
     fileout.write('\n')
fileout.close()
```



```
Riya
Jaya
Rehan
Sabu
Mabu
```

Creating a file with some names separated by newline character without using write() function

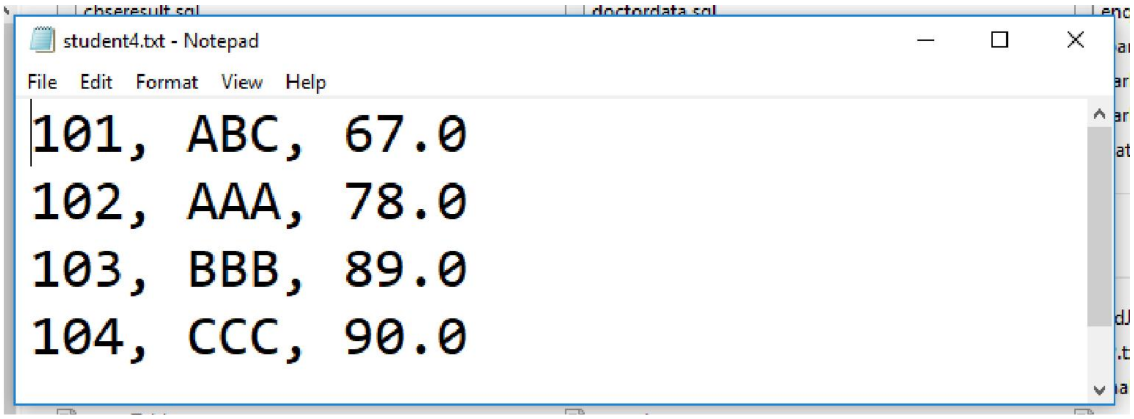
```
fileout = open("student3.txt", "w")
List1 = []
for i in range(5):
       name = input("Enter name of student :")
       List1.append(name + "\n")
fileout.writelines(List1)
fileout.close()
```



```
AAA
BBB
CCC
DDD
EEE
```

We have created comma separated values in one student record while writing in file. So we can say that the file created by program is CSV(comma separated values) format or it is a delimited file where comma is the delimiter.

```
count = int(input("How many students are there in the class"))
fileout = open("student4.txt", "w")
for i in range(count):
      print("Enter the detail for student", i+1, "below: ")
       rollno = int(input("Roll no:"))
       name = input("Name :")
       marks = float(input("Marks :"))
 rec = str(rollno) + ", " + name + ", " + str(marks) + "\n"
       fileout.write(rec)
fileout.close()
```



```
101, ABC, 67.0
102, AAA, 78.0
103, BBB, 89.0
104, CCC, 90.0
```

The flush() Function

When you write onto a file using any of the write functions, Python holds everything to write in the file in buffer and pushes it onto actual file on storage device later time. If however, you want to force Python to write the contents of buffer onto storage, you can use flush() function.

Python automatically flushes the files when closing them ie, this function is implicitly called by the close() function.

Syntax

    <fileobject>.flush()

```
f = open("out.log", "w+")
f.write('The output is \n')
```

```
f.flush()
s = 'OK'
f.write(s)
f.write('\n')
f.write('finally Over\n')
f.flush()
f.close()
```

Removing Whitespaces after Reading from file

The read() and readline() functions read data from file and return it in string form and readlines() function return entire file content in a list where each line is one item of the list.

All these functions also read the leading and trailing whitespaces ie, spaces or tabs or newline characters. If you want to remove any of the these trailing and leading whitespaces, you can use strip() functions

strip() - removes the given character from both side.
rstrip() – removes the given character from trailing end i.e. right end
lstrip() - removes the given character from leading end i.e. left end

1. Removing EOL '\n' character from the line read from the file.
   ```
   fh = file("poem.txt", "r")
   line = fh.readline()
   line = line.lstrip('\n')
   ```
2. Removing the leading whitespaces from the line read from the file.
   ```
   line = file("poem.txt", "r").readline()
   line = line.lstrip()
   ```

Handling Delimited Files

A delimited text file is a file in which the individual data values contain embedded delimiters, such as quotation marks, commas, and tabs etc.

A delimiter is a character that separates word of phrases in a line or record. There are several ways that you can process delimited data files in Python. split() is one of that.

The split() function which creates a list from a string based on a delimiter, first requires that you open a file and either read the entire contents of the file into a variable or process the file one line at a time. The function accepts a delimiter character as the argument. By default, if you leave the parameter empty, it will split a string based on space.

Consider some examples:
A file contains information in the following form:
rollno~marks1~marks2~marks3\n

12 ~ 34.50 ~ 56.75 ~ 45.50 \n

```
line = fin.readline().lstrip('\n') # remove new line character at the end
record = line.split('~')
```

The above code will create a **list** of all items in the first line

```
record = ['12', '34.50', '56.75', '45.50']
```

# split() return a list

Now you can individually access each field as list item i.e. record[0], record[1] and so on. But notice that each list item is in string form; you need to apply appropriate conversion function to get a specific type of data.