

# File Handling in Python

# Files

- Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).
- Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.
- When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.
- Hence, in Python, a file operation takes place in the following order:
  - Open a file
  - Read or write (perform operation)
  - Close the file

# Text Files and Binary Files

## Types Of File in Python

- There are two types of files in Python and each of them are explained below in detail with examples for your easy understanding.

**They are:**

- Binary file
- Text file

## Binary files in Python

- 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,** You 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.

## Binary files in Python (cont...1)

- 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.

## Text files in Python

- A text file is usually considered as sequence of lines. Line is a sequence of characters (ASCII), stored on permanent storage media. Although default character coding in python is ASCII but supports Unicode as well.
- in text file, each line is terminated by a special character, known as End of Line (EOL). From strings we know that `\n` is newline character.
- at the lowest level, text file is collection of bytes. Text files are stored in human readable form.
- they can also be created using any text editor.

## Text files in Python (Cont...1)

- 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, tex, RTF etc.
- **Tabular data:** csv, tsv etc.
- **Configuration:** ini, cfg, reg etc.

# Opening or Creating a New File in Python

- The method **open()** is used to open an existing file or creating a new file. If the complete directory is not given then the file will be created in the directory in which the python file is stored. The syntax for using open() method is given below.
  - **Syntax:**
    - `file_object = open( file_name, "Access Mode", Buffering )`
- The open method returns file object which can be stored in the name **file\_object (file-handle)**.
- **File name** is a unique name in a directory. The open() function will create the file with the specified name if it is not already exists otherwise it will open the already existing file.



# Opening Files in Python (cont...1)

- **The access mode**  
it is the string which tells in what mode the file should be opened for operations. There are three different access modes are available in python.
- **Reading:** *Reading mode is created only for reading the file. The pointer will be at the beginning of the file.*
- **Writing:** *Writing mode is used for overwriting the information on existing file.*
- **Append:** *Append mode is same as the writing mode. Instead of over writing the information this mode append the information at the end.*
- Below is the list of representation of various access modes in python.

# Opening Files in Python (cont...2)

## Access modes in Text Files

- **'r' – Read Mode:** Read mode is used only to read data from the file.
- **'w' – Write Mode:** This mode is used when you want to write data into the file or modify it. Remember write mode overwrites the data present in the file.
- **'a' – Append Mode:** Append mode is used to append data to the file. Remember data will be appended at the end of the file pointer.
- **'r+' – Read or Write Mode:** This mode is used when we want to write or read the data from the same file.
- **'a+' – Append or Read Mode:** This mode is used when we want to read data from the file or append the data into the same file.

# Opening Files in Python (cont...3)

## Access modes in Binary Files

- **'wb'** – Open a file for write only mode in the binary format.
- **'rb'** – Open a file for the read-only mode in the binary format.
- **'ab'** – Open a file for appending only mode in the binary format.
- **'rb+'** – Open a file for read and write only mode in the binary format.
- **'ab+'** – Open a file for appending and read-only mode in the binary format.

# Opening Files in Python (cont...4)

## What is Buffering ?

- Buffering is the process of storing a chunk of a file in a temporary memory until the file loads completely. In python there are different values can be given. If the buffering is set to 0 , then the buffering is off. The buffering will be set to 1 when we need to buffer the file.

# Opening Files in Python (cont...5)

## Examples Opening a file:

# open file in current directory

- `f = open("test.txt", "r")`

# specifying full path

- `f = open(r"D:\temp\data.txt", "r")`
  - -raw string
- `f = open("D:\\temp\\data.txt", "r")`
  - -absolute path

# Closing Files in Python

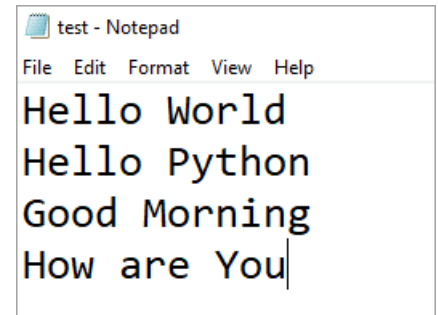
- After processing the content in a file, the file must be saved and closed. To do this we can use another method `close()` for closing the file. This is an important method to be remembered while handling files in python.
- **Syntax:** `file_object.close()`

```
string = "This is a String in Python"  
my_file = open(my_file_name.txt,"w+",1)  
my_file.write(string)  
my_file.close()  
print(my_file.closed)
```

# Reading Information in the File

- In order to read a file in python, we must open the file in read mode.
- **There are three ways in which we can read the files in python.**
  - `read([n])`
  - `readline([n])`
  - `readlines()` – all lines returned to a list
- Here, n is the number of bytes to be read.

## Reading Information in the File (cont...1)



### Example 1:

```
my_file = open("C:/Documents/Python/test.txt", "r")  
print(my_file.read(5))
```

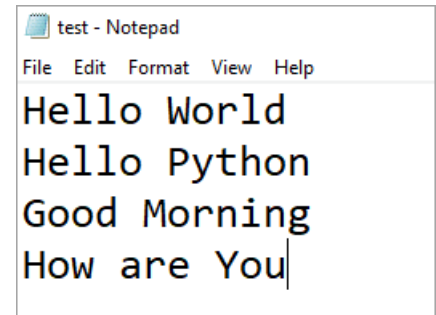
### **Output:**

Hello

- Here we are opening the file test.txt in a read-only mode and are reading only the first 5 characters of the file using the my\_file.read(5) method.



## Reading Information in the File (cont...2)



```
test - Notepad
File Edit Format View Help
Hello World
Hello Python
Good Morning
How are You|
```

### Example 2:

```
my_file = open("C:/Documents/Python/test.txt", "r")
print(my_file.read())
```

### **Output:**

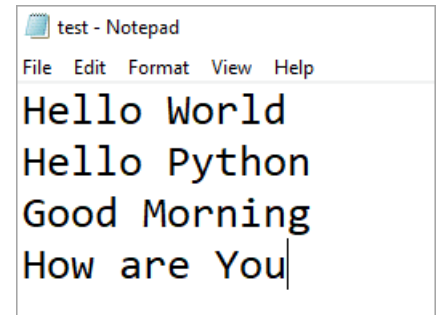
Hello World

Hello Python

Good Morning

Here we have not provided any argument inside the read() function. Hence it will read all the content present inside the file.

## Reading Information in the File (cont...3)



```
test - Notepad
File Edit Format View Help
Hello World
Hello Python
Good Morning
How are You|
```

### Example 3:

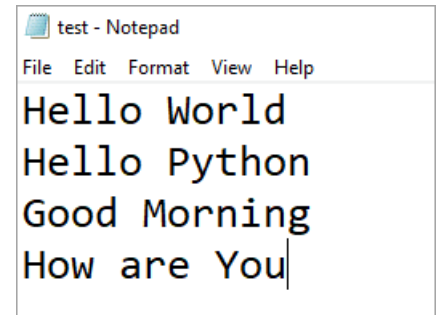
```
my_file = open("C:/Documents/Python/test.txt", "r")
print(my_file.readline(2))
```

### **Output:**

He

This function returns the first 2 characters of the next line.

## Reading Information in the File (cont...4)



```
test - Notepad
File Edit Format View Help
Hello World
Hello Python
Good Morning
How are You|
```

### Example 4:

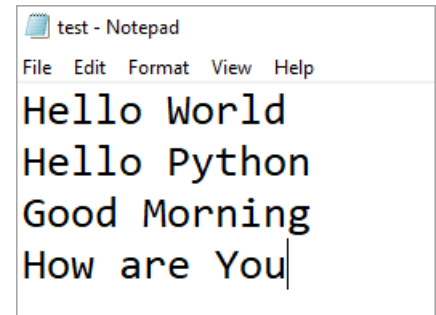
```
my_file = open("C:/Documents/Python/test.txt", "r")
print(my_file.readline())
```

### **Output:**

Hello World

Using this function we can read the content of the file on a line by line basis.

## Reading Information in the File (cont...5)



```
test - Notepad
File Edit Format View Help
Hello World
Hello Python
Good Morning
How are You|
```

### Example 5:

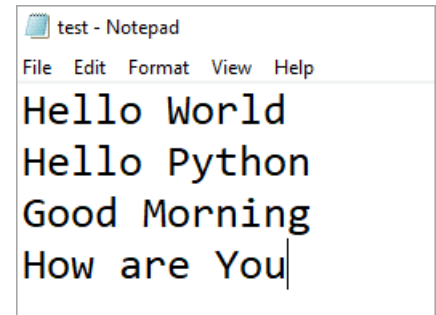
```
my_file = open("C:/Documents/Python/test.txt", "r")
print(my_file.readlines())
```

### **Output:**

```
['Hello World\n', 'Hello Python\n', 'Good Morning']
```

Here we are reading all the lines present inside the text file including the newline characters.

## Reading Information in the File (cont...6)



```
test - Notepad
File Edit Format View Help
Hello World
Hello Python
Good Morning
How are You|
```

### Reading a specific line from a File

```
line_number = 4
fo = open("C:/Documents/Python/test.txt", 'r')
currentline = 1
for line in fo:
    if(currentline == line_number):
        print(line)
        break
    currentline = currentline + 1
```

### Output:

How are You

In the above example, we are trying to read only the 4<sup>th</sup> line from the 'test.txt' file using a **“for loop”**.

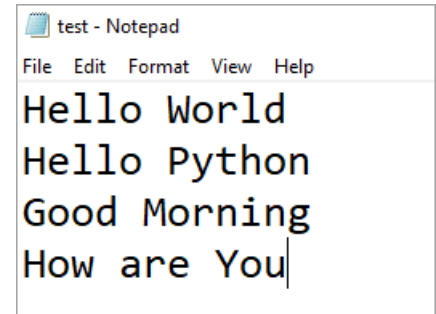
## Reading Information in the File (cont...7)

### Reading the entire file at once

```
filename = "C:/Documents/Python/test.txt"  
filehandle = open(filename, 'r')  
filedata = filehandle.read()  
print(filedata)
```

### Output:

```
Hello World  
Hello Python  
Good Morning  
How are You
```



# Write to a Python File

- In order to write data into a file, we must open the file in write mode.
- We need to be very careful while writing data into the file as it overwrites the content present inside the file that you are writing, and all the previous data will be erased.
- **We have two methods for writing data into a file as shown below.**

- write(string)

- writelines(list)

- **Example 1:**

- ```
my_file = open("C:/Documents/Python/test.txt", "w")
```

- ```
my_file.write("Hello World")
```

- The above code writes the String 'Hello World' into the 'test.txt' file.

# Write to a Python File (cont...1)

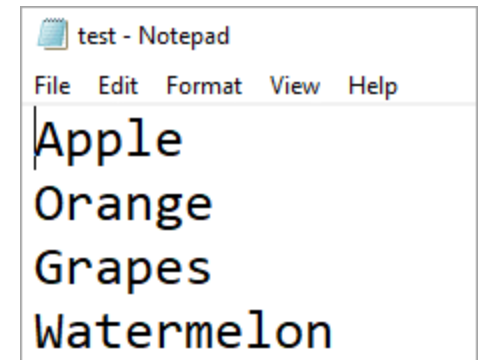
## Example 2:

```
my_file = open("C:/Documents/Python/test.txt", "w")  
my_file.write("Hello World\n")  
my_file.write("Hello Python")
```

- The first line will be 'Hello World' and as we have mentioned \n character, the cursor will move to the next line of the file and then write 'Hello Python'.
- Remember if we don't mention \n character, then the data will be written continuously in the text file like 'Hello WorldHelloPython'

## Example 3:

```
fruits = ["Apple\n", "Orange\n", "Grapes\n", "Watermelon"]  
my_file = open("C:/Documents/Python/test.txt", "w")  
my_file.writelines(fruits)
```



The above code writes a **list of data** into the 'test.txt' file simultaneously.



# Append in a Python File

To append data into a file we must open the file in 'a+' mode so that we will have access to both the append as well as write modes.

## Example 1:

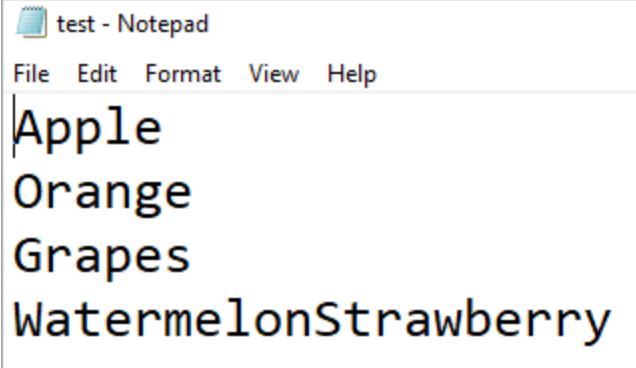
```
my_file = open("C:/Documents/Python/test.txt", "a+")  
my_file.write ("Strawberry")
```

The above code appends the string 'Strawberry' at the **end** of the 'test.txt' file

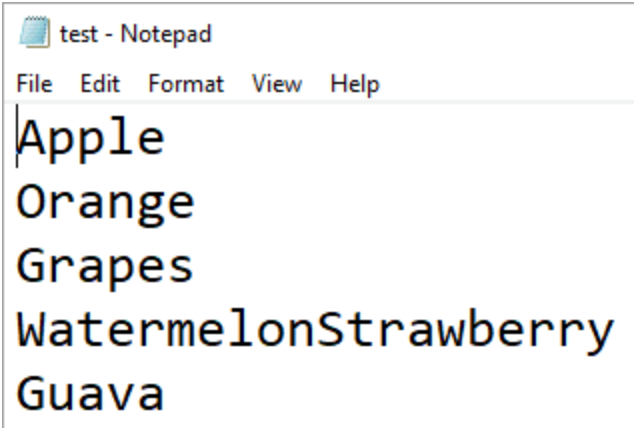
## Example 2:

```
my_file = open("C:/Documents/Python/test.txt", "a+")  
my_file.write ("\nGuava")
```

The above code appends the string 'Apple' at the end of the 'test.txt' file **in a new line**.



A screenshot of a Notepad window titled "test - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The text area contains the following text on four lines: "Apple", "Orange", "Grapes", and "WatermelonStrawberry".



A screenshot of a Notepad window titled "test - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The text area contains the following text on five lines: "Apple", "Orange", "Grapes", "WatermelonStrawberry", and "Guava".

# `flush()` function

- When we write any data to file, python hold everything in buffer (temporary memory) and pushes it onto actual file later. If you want to force Python to write the content of buffer onto storage, you can use `flush()` function.
- Python automatically flushes the files when closing them i.e. it will be implicitly called by the `close()`, BUT if you want to flush before closing any file you can use `flush()`

# flush() function...cont..1

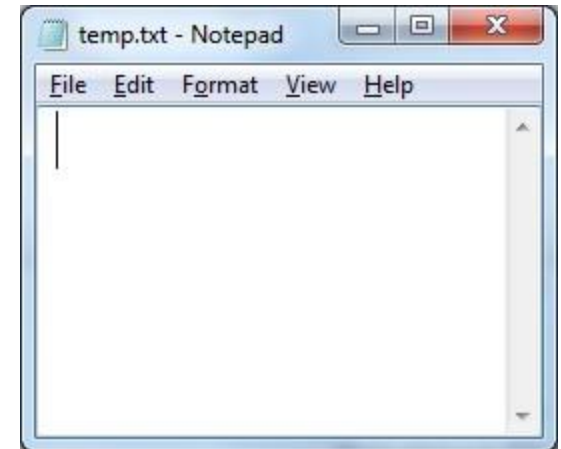
Example: working of flush()

## Without flush()

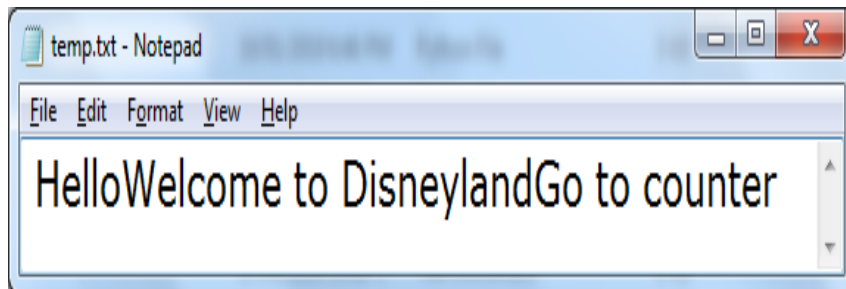
```
myfile = open("temp.txt", "w+")  
myfile.write("Hello")  
myfile.write("Welcome to Disneyland")  
n = input("press any key")  
myfile.write("Go to counter")  
myfile.close()
```

When you run the above code, program will stopped at "Press any key", for time being don't press any key and go to folder where file "temp.txt" is created an open it to see what is in the file till now

Nothing is in the file temp.txt



**NOW PRESS ANY KEY....**



Now content is stored, because of close() function contents are flushed and pushed in file

# flush() function...cont..2

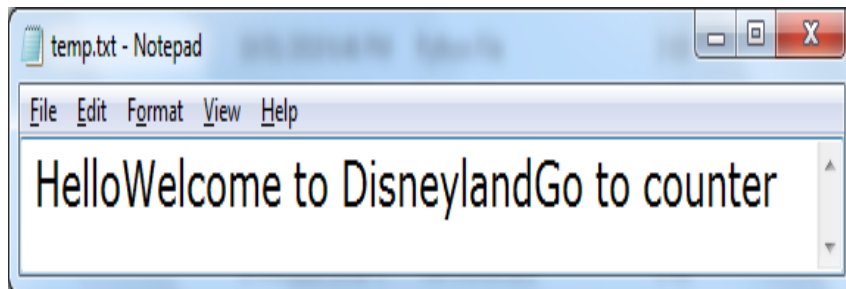
Example: working of flush()

## With flush()

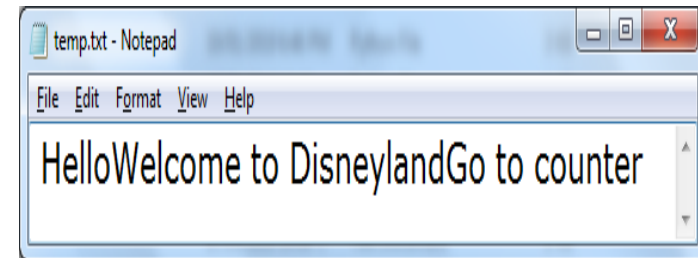
```
myfile = open("temp.txt", "w+")  
myfile.write("Hello")  
myfile.write("Welcome to Disneyland")  
myfile.flush()  
n = input("press any key")  
myfile.write("Go to counter")  
myfile.close()
```

When you run the above code, program will stopped at "Press any key", for time being don't press any key and go to folder where file "temp.txt" is created an open it to see what is in the file till now

**NOW PRESS ANY KEY....**



All contents before flush() are present in file



Rest of the content is written because of close(), contents are flushed and pushed in file.

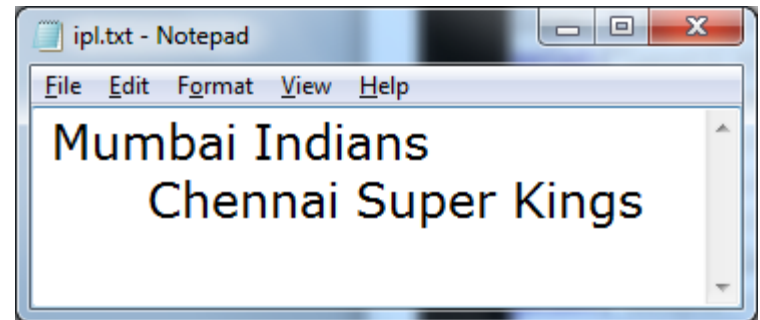
# Removing whitespaces after reading from file

- `read()` and `readline()` reads data from file and return it in the form of string and `readlines()` returns data in the form of list.
- All these read function also read leading and trailing whitespaces, new line characters. If you want to remove these characters you can use functions
  - `strip()` : removes the given character from both ends.
  - `lstrip()`: removes given character from left end
  - `rstrip()`: removes given character from right end

# Removing whitespaces after reading from file cont...1

- Example: strip(), lstrip(), rstrip()

```
myfile = open("ipl.txt")
line1 = myfile.readline()
print("Length of Line is :",len(line1))
line1 = line1.rstrip('\n')
print("Length of Line is :",len(line1))
line2 = myfile.readline()
print("Length of Line is :",len(line2))
line2 = line2.lstrip()
print("Length of Line is :",len(line2))
```



```
Length of Line is : 15
Length of Line is : 14
Length of Line is : 25
Length of Line is : 19
```

# File Pointer

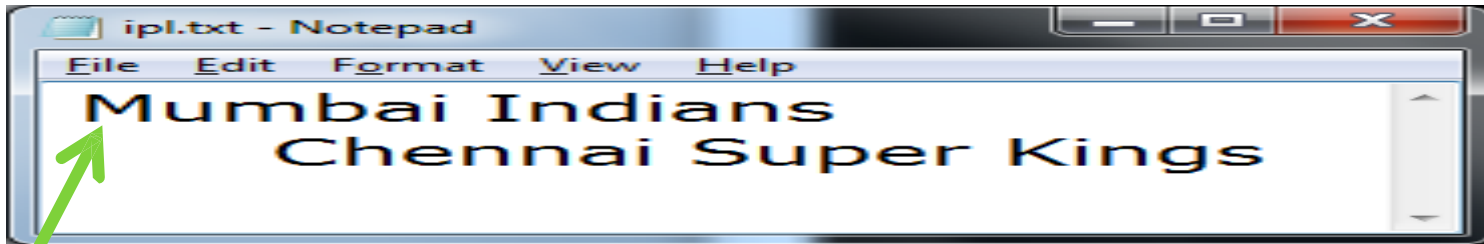
- Every file maintains a file pointer which tells the current position in the file where reading and writing operation will take.
- When we perform any read/write operation two things happens:
  - The operation at the current position of file pointer
  - File pointer advances by the specified number of bytes.

# File Pointer

cont...1

## Example

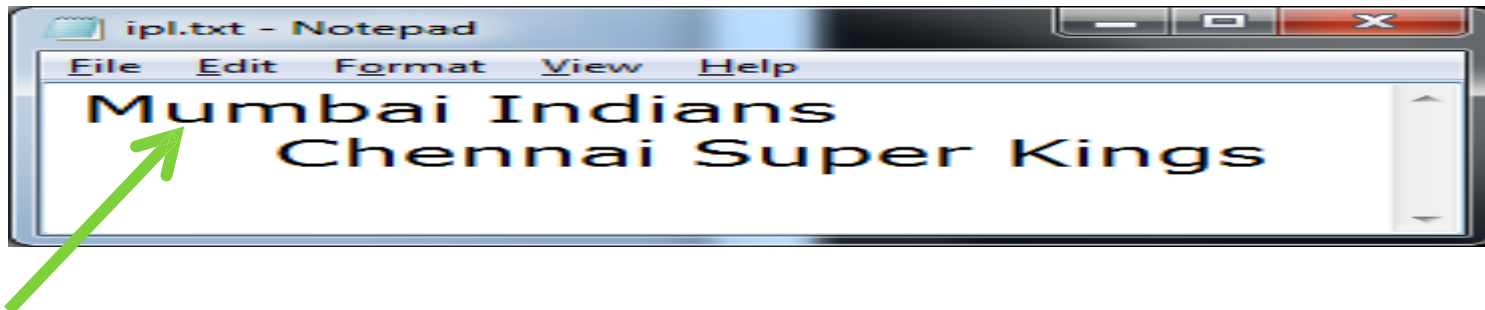
```
myfile = open("ipl.txt","r")
```



File pointer will be by default at first position i.e. first character

```
ch = myfile.read(1)
```

ch will store first character i.e. first character is consumed, and file pointer will move to next character





# Binary file operations

- If we want to write a structure such as list or dictionary to a file and read it subsequently we need to use the **Python module pickle**. Pickling is the process of converting structure to a byte stream before writing to a file and while reading the content of file a reverse process called Unpickling is used to convert the byte stream back to the original format.

# Binary file operations

cont... 1

- First we need to import the module called pickle.
- This module provides 2 main functions:
  - dump() : to write the object in file which is loaded in binary mode
    - **Syntax :**     **dump(object\_to\_write, filehandle)**
  - load() : dumped data can be read from file using load() i.e. it is used to read object from pickle file.
    - **Syntax:**     **object = load(filehandle)**

# Binary file operations

cont... 2

- Example: dump()

```
# program to demonstrate binary file operations
import pickle
myfile = open("project.txt", "wb")
dict1 = {'ename': 'jitendra', 'pname': 'simulator', 'charge': 45000}
pickle.dump(dict1, myfile)
myfile.close()
```



See the content is some kind of encrypted format, and it is not in complete readable form

# Binary file operations

cont... 3

- **Example: load()**

```
# program to demonstrate binary file operations
import pickle
mf = open("project.txt","wb")
#dict1 = pickle.load(mf)
#myfile.close()
dict1 = {1:"a",2:"b",3:"c"}
pickle.dump(dict1,mf)
mf.close()
```

```
# program to demonstrate binary file operations
import pickle
mf = open("project.txt","rb")
dict1 = pickle.load(mf)
print(dict1)
print("Item 1 is ",dict1[1])
mf.close()
```

```
{1: 'a', 2: 'b', 3: 'c'}
Item 1 is a
```

# Binary file operations

cont... 4

The four major operations performed using a binary file are—

- 1. Inserting/Appending a record in a binary file
- 2. Reading records from a binary file
- 3. Searching a record in a binary file
- 4. Updating a record in a binary file

# Binary file operations

cont... 5

- **Inserting/Appending a record in a binary file**
- Inserting or adding (appending) a record into a binary file requires importing pickle module into a program followed by dump() method to write onto the file.

# Binary file operations

cont... 5a

- Inserting/Appending a record in a binary file

```
021_01_BinaryFileInsert.py - D:\Books_Tutorials\KV Academics\CS_XII_CBSE_2020-21\Class-XII\021_01_Bina...
File Edit Format Run Options Window Help
#021_01_BinaryFileInsert.py
#Program to insert/append a record in the binary file

import pickle
record = []

while True:
    roll_no = int (input("Enter Student Roll No: "))
    name = input("Enter Student Name: ")
    marks = int (input("Enter Student Marks: "))
    data = [roll_no, name,marks]
    record.append(data)
    choice = input("Do you want to add more records: (Y/N)? ")
    if choice.upper()=='N':
        break

f = open("D:\\datafile\\Student.dat", "wb")

pickle.dump(record, f)
print("Records Added")
f.close()
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018,
16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for
more information.
>>>
RESTART: D:\Books_Tutorials\KV Academics\CS_X
II_CBSE_2020-21\Class-XII\021_01_BinaryFileIns
ert.py
Enter Student Roll No: 1
Enter Student Name: ramesh
Enter Student Marks: 45
Do you want to add more records: (Y/N)? :y
Enter Student Roll No: 2
Enter Student Name: suresh
Enter Student Marks: 78
Do you want to add more records: (Y/N)? :n
Records Added
>>>
```

```
Student.dat - Notepad
File Edit Format View Help
[?]q ([?]q[?]K[?]X[?] rameshq K-e]q[?]K X[?] sureshq[?]KNe.
```

- **Reading a record from a binary file**
- deals with reading the contents from binary file student using load() method of pickle module. It is used to read the object from the opened file. The syntax for this is given by the statement—
  - object = pickle.load(file)



# Binary file operations

cont... 5c

- Reading a record from a binary file

```
021_02_BinaryFileReadpy.py - D:/Books_Tutorials/KV Academics/CS_XII_CBSE_2020-21/Class-XII/021_0...
File Edit Format Run Options Window Help
#021_02_BinaryFileReadt.py
#Program to Read a record in the binary
#file "D:\\datafile\\Student.dat".

import pickle
f = open("D:\\datafile\\Student.dat", "rb")
stud_rec = pickle.load(f) # To read the obje
print("Content of student file are: ")

#Reading the fields from the file
for R in stud_rec:
    roll_no = R[0]
    name = R[1]
    marks = R[2]
    print(roll_no, name, marks)

f.close()

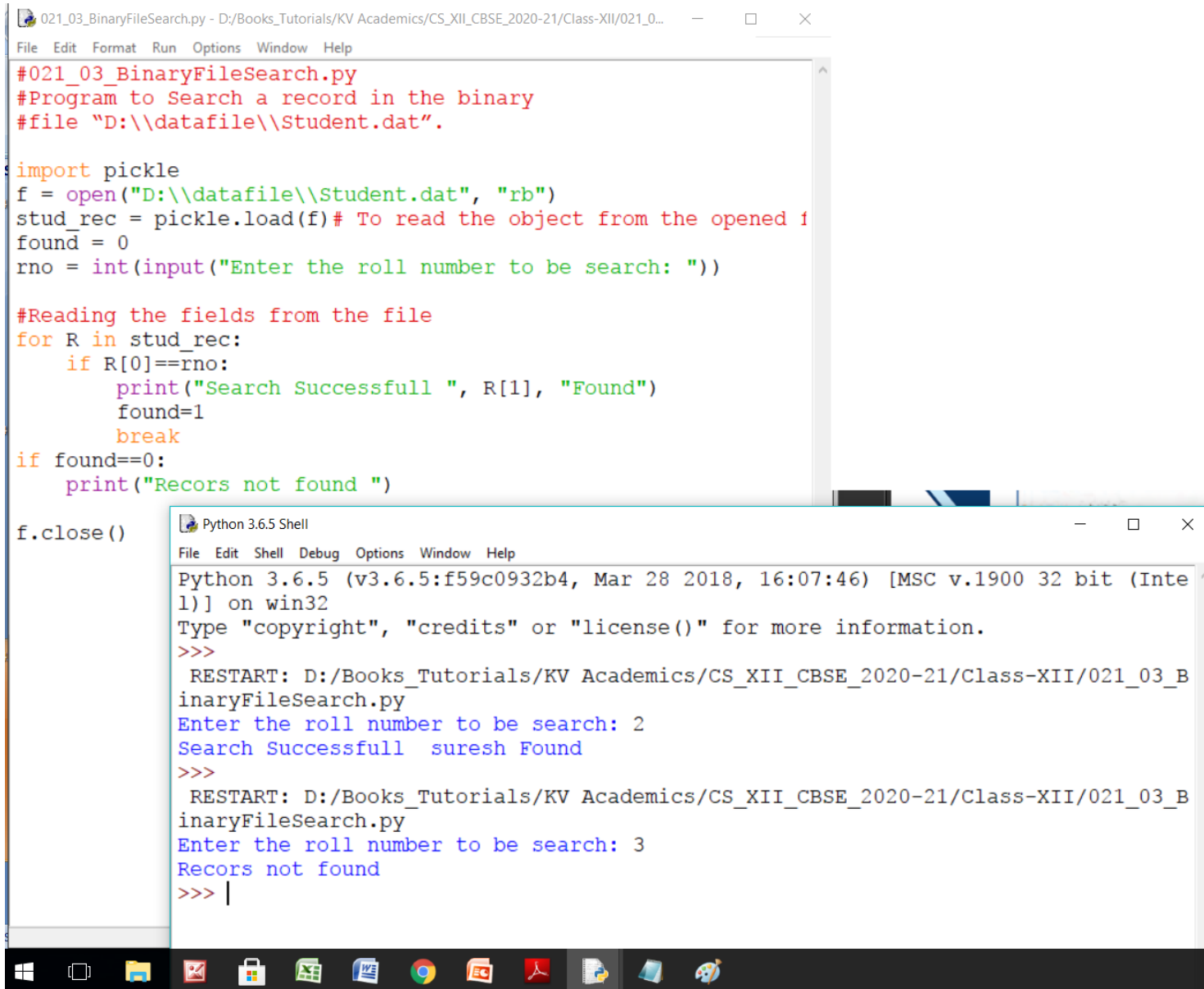
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16
:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for m
ore information.
>>>
RESTART: D:/Books_Tutorials/KV Academics/CS_XII
_CBSE_2020-21/Class-XII/021_02_BinaryFileReadpy.
py
Content of student file are:
1 ramesh 45
2 suresh 78
>>>
```

- **Searching a record in a binary file**
- Searching the binary file ( "student" ) is carried out on the basis of the roll number entered by the user. The file is opened in the read-binary mode and gets stored in the file object, f. load() method is used to read the object from the opened file. A variable 'found' is used which will tell the status of the search operation being successful or unsuccessful. Each record from the file is read and the content of the field, roll no, is compared with the roll number to be searched. Upon the search being successful, appropriate message is displayed to the user.

# Binary file operations

cont... 5e

- Searching a record in a binary file



```
#021_03_BinaryFileSearch.py
#Program to Search a record in the binary
#file "D:\\datafile\\Student.dat".

import pickle
f = open("D:\\datafile\\Student.dat", "rb")
stud_rec = pickle.load(f)# To read the object from the opened file
found = 0
rno = int(input("Enter the roll number to be search: "))

#Reading the fields from the file
for R in stud_rec:
    if R[0]==rno:
        print("Search Successfull ", R[1], "Found")
        found=1
        break
if found==0:
    print("Recors not found ")

f.close()
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: D:/Books_Tutorials/KV Academics/CS_XII_CBSE_2020-21/Class-XII/021_03_BinaryFileSearch.py
Enter the roll number to be search: 2
Search Successfull suresh Found
>>>
RESTART: D:/Books_Tutorials/KV Academics/CS_XII_CBSE_2020-21/Class-XII/021_03_BinaryFileSearch.py
Enter the roll number to be search: 3
Recors not found
>>> |
```

- **Updating a record in a binary file**
- Updating a record in the file requires roll number (search field) to be fetched from the user whose name (Record) is to be updated
- Once the record is found, the file pointer is moved to the beginning of the file using seek(0) statement, and then the changed values are written to the file and the record is updated. seek() method is used for random access to the file. (more about seek() in later sessions)

# Binary file operations

cont... 5g

- Updating a record in a binary file

```
021_04_BinaryFileUpdate.py - D:/Books_Tutorials/KV Academics/CS_XII_CBSE_2020-21/Class-XII/021_04_BinaryFil...
File Edit Format Run Options Window Help
#021_04_BinaryFileUpdate.py
#Program to update a record in the binary
#file "D:\\datafile\\Student.dat".

import pickle
f = open("D:\\datafile\\Student.dat", "rb+")
stud_rec = pickle.load(f)# To read the object from the opened file
found = 0
rno = int(input("Enter the roll number to be search: "))

#Reading the fields from the file
for R in stud_rec:
    if R[0]==rno:
        print("Current Values are ",R[1], R[2])
        R[1] = input("Enter New Name: ")
        R[2] = int(input("Enter New Marks: "))
        found=1
        break
if found==1:
    f.seek(0) # Placing file pointer to the begining of the file
    pickle.dump(stud_rec,f)
    print("Recors updated! ")
else:
    print("Record not found")

f.close()
```

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900
1)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: D:/Books_Tutorials/KV Academics/CS_XII_CBSE_2020-21/Class-
inaryFileUpdate.py
Enter the roll number to be search: 1
Current Values are ramesh 45
Enter New Name: Mukesh
Enter New Marks: 55
Recors updated!
>>>
RESTART: D:/Books_Tutorials/KV Academics/CS_XII_CBSE_2020-21/Class-
inaryFileUpdate.py
Enter the roll number to be search: 3
Record not found
>>>
```

- **RANDOM ACCESS IN FILES USING TELL() AND SEEK()**
- Till now, in all our programs we laid stress on the sequential processing of data in a text and binary file.
- But files in Python allow random access of the data as well using built-in methods seek() and tell().

- RANDOM ACCESS IN FILES USING TELL() AND SEEK()
- **seek()**—**seek()** function is used to change the position of the file handle (file pointer) to a given specific position. File pointer is like a cursor, which defines from where the data has to be read or written in the file.
- Python file method `seek()` sets the file's current position at the offset. This argument is optional and defaults to 0, which means absolute file positioning. Other values are: 1, which signifies seek is relative (may change) to the current position, and 2, which means seek is relative to the end of file. There is no return value.
- The reference point is defined by the “`from_what`” argument. It can have any of the three values:
  - 0: sets the reference point at the beginning of the file, which is by default.
  - 1: sets the reference point at the current file position.
  - 2: sets the reference point at the end of the file.

- RANDOM ACCESS IN FILES USING TELL() AND SEEK()
- seek() can be done in two ways:
  - Absolute Positioning
  - Relative Positioning
- Absolute referencing using seek() gives the file number on which the file pointer has to position itself. The syntax for seek() is—
  - f.seek(file\_location) #where f is the file pointer
- *For example, f.seek(20) will give the position or file number where the file pointer has been placed. This statement shall move the file pointer to 20th byte in the file no matter where you are.*



- RANDOM ACCESS IN FILES USING TELL() AND SEEK()
- Relative referencing/positioning has two arguments, offset and the position from which it has to traverse. The syntax for relative referencing is:
  - `f.seek(offset, from_what)` #where f is file pointer, *For example,*
  - `f.seek(-10,1)` from current position, move 10 bytes backward
  - `f.seek(10,1)` from current position, move 10 bytes forward
  - `f.seek(-20,1)` from current position, move 20 bytes backward
  - `f.seek(10,0)` from beginning of file, move 10 bytes forward

# Binary file operations

cont... 6b

- **RANDOM ACCESS IN FILES USING TELL() AND SEEK()**
- **tell()—tell() returns the current position of the file read/write pointer within the file. Its syntax is:**
- **f.tell() #where f is file pointer**
- When we open a file in reading/writing mode, the file pointer rests at 0<sup>th</sup> byte.
- When we open a file in append mode, the file pointer rests at the last byte.
- This is illustrated in the practical implementation that follows:

# Binary file operations

cont... 7

- **RANDOM ACCESS IN FILES USING TELL() AND SEEK()**

021\_05\_BinaryFileSeekTell.py - D:/Books\_Tutorials/KV Academics/CS\_XII\_CBSE\_2020-21/Class-XII/021\_05\_BinaryFi...

```
File Edit Format Run Options Window Help
#021_05_BinaryFileSeekTell.py
#Program to illustrate seek() and tell()
#file "D:\\datafile\\test.txt".

f = open("D:\\datafile\\test.txt", "r")

print("Before reading, File Pointer Position : ", f.tell())

s = f.read()
print("After reading, File Pointer Position : ", f.tell())

f.seek(0) #Brings the file pointer to the 0th byte

print("From the beginning of the file again: ", f.tell())
s = f.read(4)
print("First 4 bytes are: ", s)
print("File Pointer Position : ", f.tell())

s = f.read(3)
print("Next 4 bytes are: ", s)
print("File Pointer Position : ", f.tell())

f.close()
```

Python 3.6.5 Shell

```
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:4
l)] on win32
Type "copyright", "credits" or "license()" for more i
>>>
RESTART: D:/Books_Tutorials/KV Academics/CS_XII_CBSE
inaryFileSeekTell.py
Before reading, File Pointer Position : 0
After reading, File Pointer Position : 79
From the beginning of the file again: 0
First 4 bytes are: Indi
File Pointer Position : 4
Next 4 bytes are: a i
File Pointer Position : 7
>>>
```

test.txt - Notepad

```
File Edit Format View Help
India is my country
All indians are my brothers and sisters
I love my country
```

# CSV File operations in Python

- A CSV file (Comma Separated Values file) is a type of plain text file that uses specific structuring to arrange tabular data. Because it's a plain text file, it can contain only actual text data—in other words, printable [ASCII](#) or [Unicode](#) characters.
- The structure of a CSV file is given away by its name. Normally, CSV files use a comma to separate each specific data value. Here's what that structure looks like:

## CSV

```
column 1 name,column 2 name, column 3 name
first row data 1,first row data 2,first row data 3
second row data 1,second row data 2,second row data 3
...
```

# CSV File operations in Python

Cont...01

- Normally, the first line identifies each piece of data—in other words, the name of a data column. Every subsequent line after that is actual data and is limited only by file size constraints.
- In general, the separator character is called a delimiter, and the comma is not the only one used. Other popular delimiters include the tab (\t), colon (:), and semi-colon (;) characters. Properly parsing a CSV file requires us to know which delimiter is being used.
- CSV is a simple flat file in a human readable format which is extensively used to store tabular data, in a spreadsheet or database. A CSV file stores tabular data (numbers and text) in plain text.

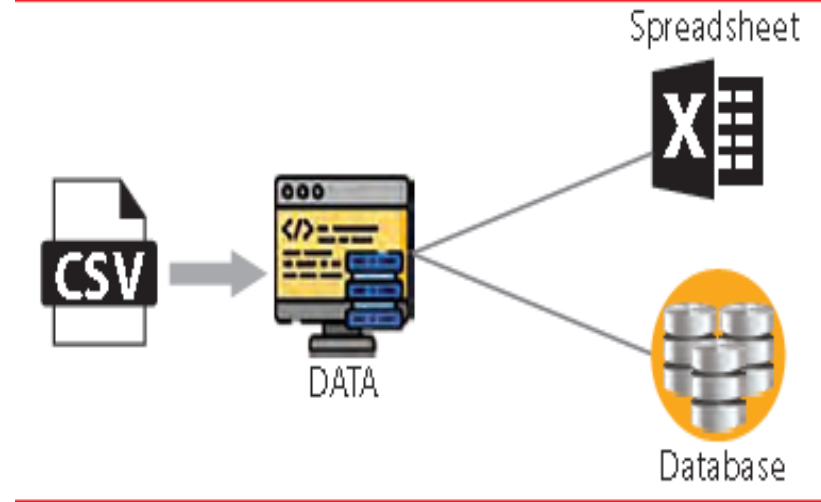
## CSV

```
column 1 name,column 2 name, column 3 name  
first row data 1,first row data 2,first row data 3  
second row data 1,second row data 2,second row data 3  
...
```

# CSV File operations in Python

Cont...02

- Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel or OpenOffice Calc.
- CSV stands for “**comma separated values**”. **Thus, we can say that a comma separated file is a delimited text file that uses a comma to separate values.**



Each line in a file is known as **data/record**. Each record consists of one or more fields, separated by commas (also known as delimiters), *i.e., each of the records is also a part of this file. Tabular data is stored as text in a CSV file. The use of comma as a field separator is the source of the name for this file format. It stores our data into a spreadsheet or a database.*



- **WHY USE CSV?**
- The extensive use of social networking sites and their various associated applications requires the handling of huge data. But the problem arises as to how to handle and organize this large unstructured data?
- The solution to the above problem is CSV. Thus, CSV organizes data into a structured form and, hence, the proper and systematic organization of this large amount of data is done by CSV. Since CSV file formats are of plain text format, it makes it very easy for website developers to create applications that implement CSV.
- the several advantages that are offered by CSV files are as follows:
  - CSV is faster to handle.
  - CSV is smaller in size.
  - CSV is easy to generate and import onto a spreadsheet or database.
  - CSV is human readable and easy to edit manually.
  - CSV is simple to implement and parse.
  - CSV is processed by almost all existing applications.

- For working with CSV files in Python, there is an inbuilt module called **CSV**. **It is used to read and write tabular data in CSV format.**
- To perform read and write operations with CSV file, we must import **CSV module**. **CSV module can handle CSV files correctly regardless of the operating system on which the files were created.**
- Along with this module, `open()` function is used to open a CSV file and return file object. We load the module in the usual way using `import`:
  - `>>> import csv`
- Like other files (text and binary) in Python, there are two basic operations that can be carried out on a CSV file:
  - 1. Reading from a CSV file
  - 2. Writing to a CSV file

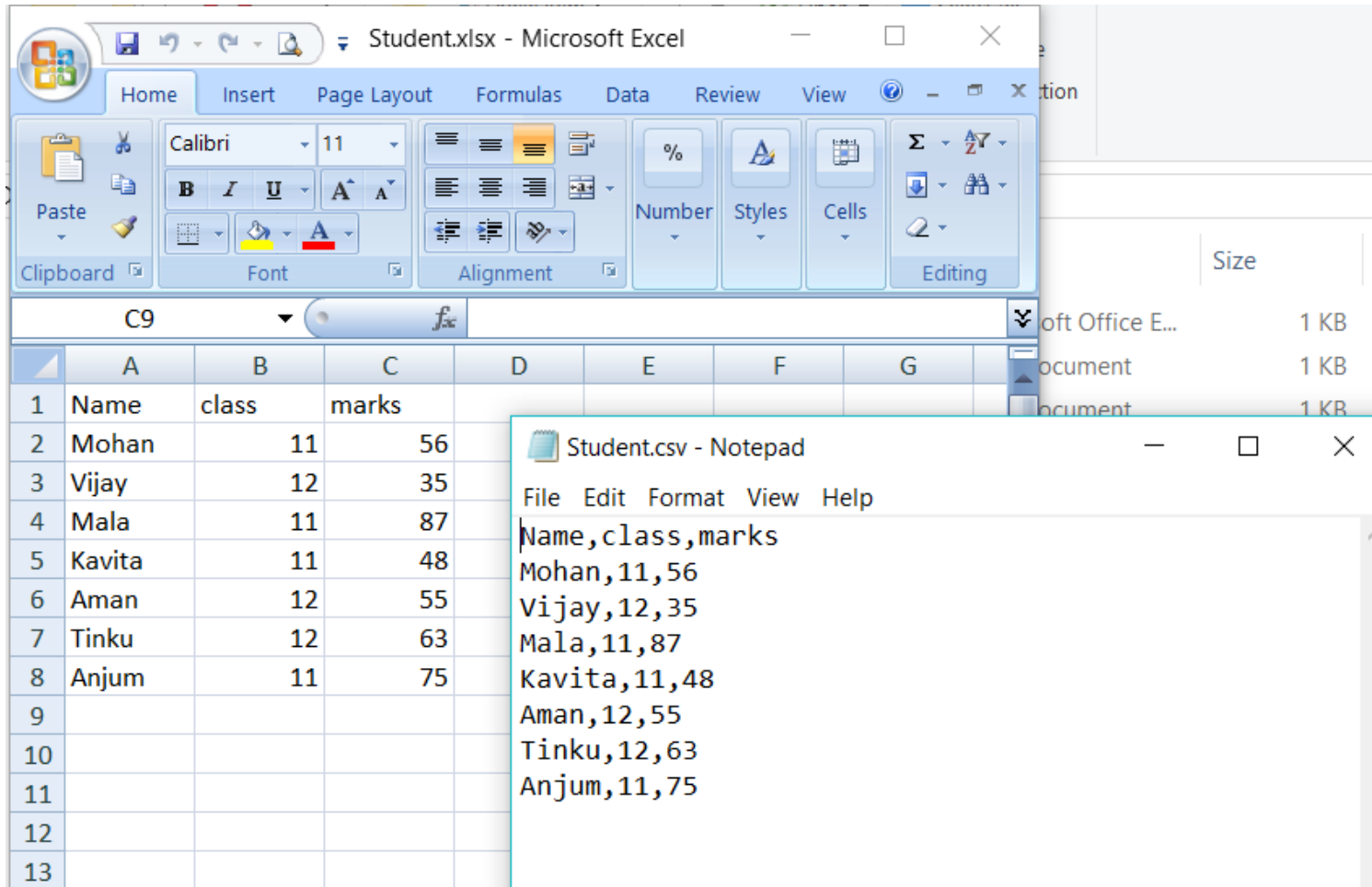


- For working with CSV files in Python, there is an inbuilt module called **CSV**. **It is used to read and write tabular data in CSV format.**
- To perform read and write operations with CSV file, we must import **CSV module**. **CSV module can handle CSV files correctly regardless of the operating system on which the files were created.**
- Along with this module, `open()` function is used to open a CSV file and return file object. We load the module in the usual way using `import`:
  - `>>> import csv`
- Like other files (text and binary) in Python, there are two basic operations that can be carried out on a CSV file:
  - 1. Reading from a CSV file
  - 2. Writing to a CSV file

- **Reading from a CSV File**
- Reading from a CSV file is done using the reader object. The CSV file is opened as a text file with Python's built-in `open()` function, which returns a file object. This creates a special type of object to access the CSV file (reader object), using the `reader()` function.
- The reader object is an iterable that gives us access to each line of the CSV file as a list of fields. We can also use `next()` directly on it to read the next line of the CSV file, or we can treat it like a list in a for loop to read all the lines of the file (as lists of the file's fields).
- Let us enter the student details in spreadsheet and save this file as shown.
- Next step is to open the Notepad and enter the data for `student.csv`, which will be the equivalent for `student.xls`.

# CSV File operations in Python

Cont...04\_a



The image shows a screenshot of a Microsoft Excel spreadsheet titled 'Student.xlsx' and a Notepad window titled 'Student.csv'. The Excel spreadsheet has columns labeled 'Name', 'class', and 'marks' in the first row. The data rows are as follows:

	A	B	C	D	E	F	G
1	Name	class	marks				
2	Mohan	11	56				
3	Vijay	12	35				
4	Mala	11	87				
5	Kavita	11	48				
6	Aman	12	55				
7	Tinku	12	63				
8	Anjum	11	75				
9							
10							
11							
12							
13							

The Notepad window shows the CSV file content:

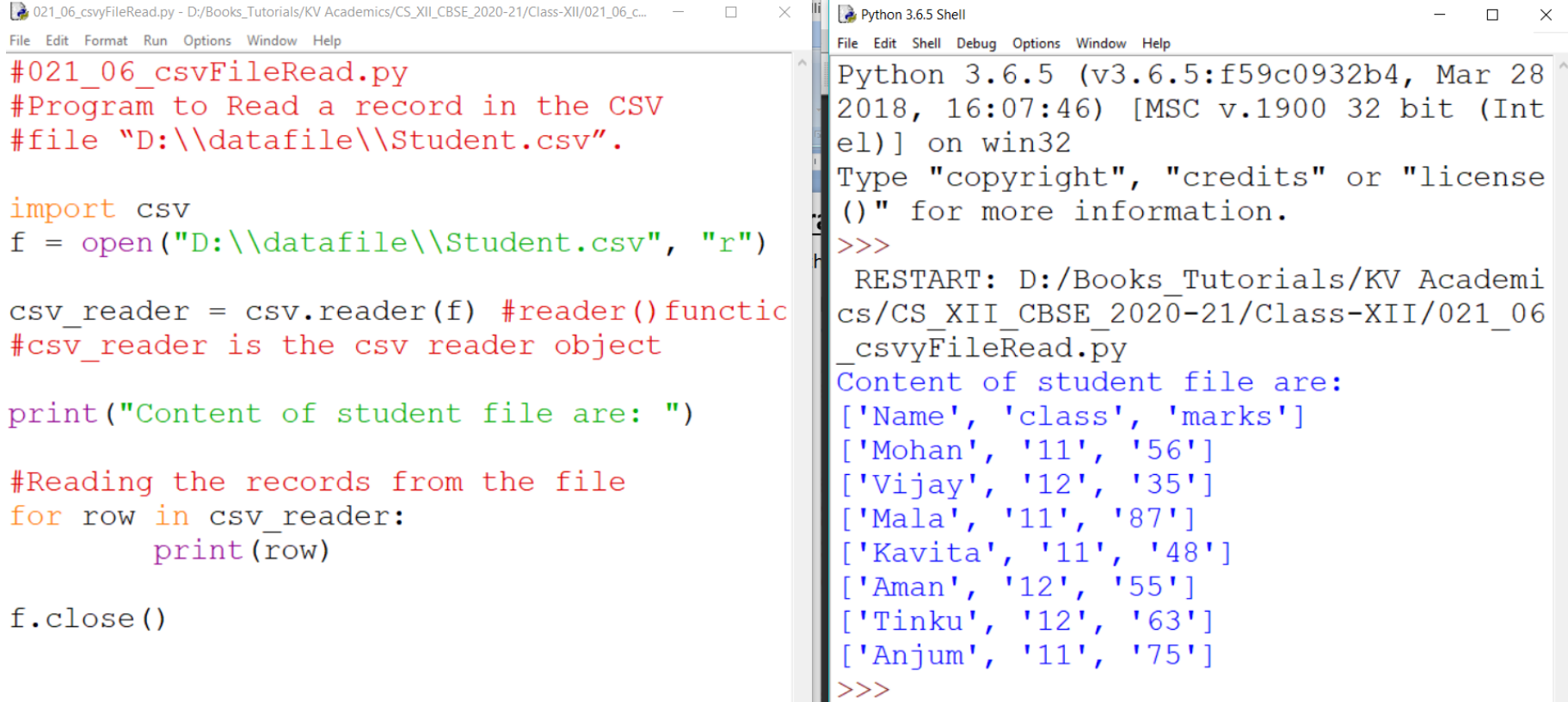
```
Student.csv - Notepad
File Edit Format View Help
Name,class,marks
Mohan,11,56
Vijay,12,35
Mala,11,87
Kavita,11,48
Aman,12,55
Tinku,12,63
Anjum,11,75
```

In student.csv (notepad) file, the first line is the header and remaining lines are the data/records. The fields are separated by comma. In general, the separator character is called a delimiter, and the comma is not the only one used. Other popular delimiters include the tab (\t), colon (:), and semi-colon (;) characters.

# CSV File operations in Python

Cont...04\_b

Program to read the contents of "student.csv" file



```
#021_06_csvFileRead.py
#Program to Read a record in the CSV
#file "D:\\datafile\\Student.csv".

import csv
f = open("D:\\datafile\\Student.csv", "r")

csv_reader = csv.reader(f) #reader() function
#csv_reader is the csv reader object

print("Content of student file are: ")

#Reading the records from the file
for row in csv_reader:
    print(row)

f.close()
```

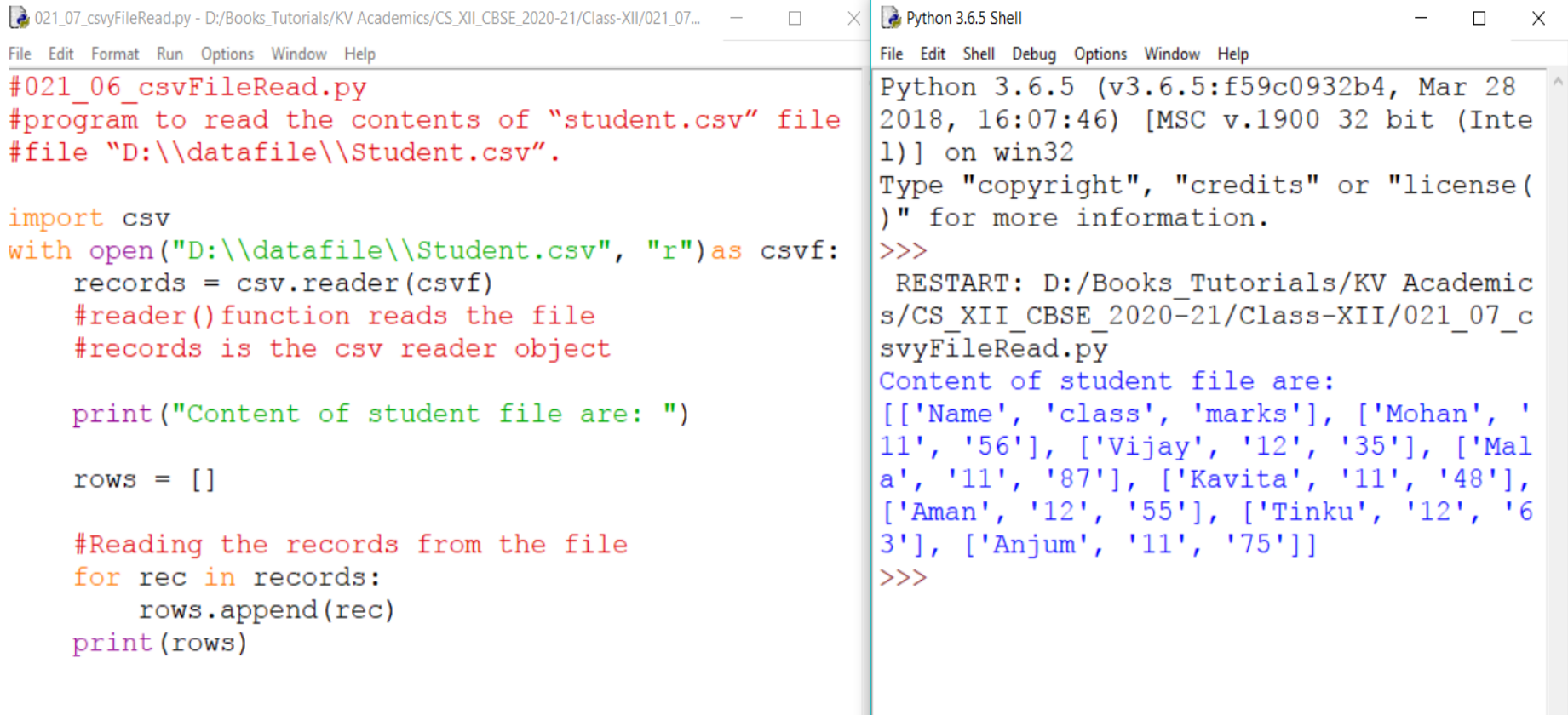
```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28
2018, 16:07:46) [MSC v.1900 32 bit (Int
el)] on win32
Type "copyright", "credits" or "license
()" for more information.
>>>
RESTART: D:/Books_Tutorials/KV Academi
cs/CS_XII_CBSE_2020-21/Class-XII/021_06
_csvyFileRead.py
Content of student file are:
['Name', 'class', 'marks']
['Mohan', '11', '56']
['Vijay', '12', '35']
['Mala', '11', '87']
['Kavita', '11', '48']
['Aman', '12', '55']
['Tinku', '12', '63']
['Anjum', '11', '75']
>>>
```

Every record is stored in reader object in the form of a List. We first open the CSV file in READ mode. The file object is named f. The file object is converted to csv.reader object. The reader object is used to read records as lists from a csv file. Iterate through all the rows using a for loop. row is nothing but a list containing all the field values

# CSV File operations in Python

Cont...04\_c

Read the contents of “student.csv” file using with open().



```
021_07_csvyFileRead.py - D:/Books_Tutorials/KV Academics/CS_XII_CBSE_2020-21/Class-XII/021_07... Python 3.6.5 Shell
File Edit Format Run Options Window Help
#021_06_csvFileRead.py
#program to read the contents of "student.csv" file
#file "D:\\datafile\\Student.csv".

import csv
with open("D:\\datafile\\Student.csv", "r") as csvf:
    records = csv.reader(csvf)
    #reader()function reads the file
    #records is the csv reader object

    print("Content of student file are: ")

    rows = []

    #Reading the records from the file
    for rec in records:
        rows.append(rec)
    print(rows)

Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: D:/Books_Tutorials/KV Academics/CS_XII_CBSE_2020-21/Class-XII/021_07_csvyFileRead.py
Content of student file are:
[['Name', 'class', 'marks'], ['Mohan', '11', '56'], ['Vijay', '12', '35'], ['Malika', '11', '87'], ['Kavita', '11', '48'], ['Aman', '12', '55'], ['Tinku', '12', '63'], ['Anjum', '11', '75']]
>>>
```

Code uses “with open()” function, the only difference being that the file being opened using with open() gets automatically closed after the program execution gets over, unlike open() where we need to give close() statement explicitly.