

# DATA FILES IN PYTHON



## WHY DATA FILES ??????????????????

As we know whenever we enter data while running programs, it is not saved anywhere and we have to enter it again when we run the program again. So to store required data permanently on hard disk (Secondary Storage Device) we need to store it in File.

**Note : File is a Stream or sequence of bytes /characters**

Python Data Files can be of two types

1. Text File (By default it creates file in **text Mode**)
2. Binary File

Difference between Text and Binary Files

| S.No. | Python Text File   | Python Binary Files  |
|-------|--|--|
| 1.    | Consists Data in ASCII (Human readable form).  | Consists Data in Binary form   |
| 2.    | Suitable to store Unicode characters also  | Suitable to store binary data such as images, video files , audio files etc. |
| 3.    | Each line in Text file is terminated with a Special character EOL(end of line)             | There is no EOL character  |
| 4.    | Operation on text files are slower than binary files as data is to be translated to binary | Operation on binary files are faster as no translation required              |



### Operations on File:

1. How to Handle Data File/ Text File:

Whenever we worked with Data File in Python we have to follow sequence

- 1.1 Open/Create File
- 1.2 Read from/Write to file
- 1.3 Close File

## We can do following tasks/operations with python Data File.

- a) Creation/Opening of an existing Data File
- b) Reading from file
- c) Writing to Data File
- d) Appending data ( inserting Data at the end of the file)
- e) Inserting data (in between the file)
- f) Deleting Data from file
- g) Copying a file
- h) Modification/Updation in Data File.

### Functions Used for File Handling

| s.no | Function Name | Syntax                                | Use   |
|------|---------------|---------------------------------------|---|
| 1    | open()        | F_obj=open("File_name", mode)         | To Open or create a file in desired mode                    |
| 2    | close         | F_obj.close()                         | To Close the file   |
| 3    | read()        | F_obj.read() or F_obj.read(n)         | To read all or specified no. of characters from file        |
| 4    | readline()    | F_obj.readline() or F_obj.readline(n) | To read a single line or specified no. of characters        |
| 5    | readlines()   | F_obj.readlines()                     | To read all lines from a file and returns it in the form of |
| 6    | write()       | F_obj.write(str)                      | To write data (of string type) on to the file               |
| 7    | writelines()  | F_obj.writelines(LST)                 | To Write Sequence (list/tuple etc) of strings in a file     |



Before discussing reading and writing operation let's understand the **File Modes**. Consider the table of modes given below

Here is a list of the different modes of opening a file –

| Sr.No. | Modes & Description  |
|--------|--|
| 1      | <b>r</b><br>Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.  |
| 2      | <b>rb</b><br>Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.  |
| 3      | <b>r+</b><br>Opens a file for both reading and writing. The file pointer placed at the beginning of the file.  |
| 4      | <b>rb+</b><br>Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.  |
| 5      | <b>w</b><br>Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.   |
| 6      | <b>wb</b><br>Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.   |
| 7      | <b>w+</b><br>Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.   |
| 8      | <b>wb+</b><br>Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.   |
| 9      | <b>a</b><br>Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.   |
| 10     | <b>ab</b><br>Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.                         |
| 11     | <b>a+</b><br>Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.                   |
| 12     | <b>ab+</b><br>Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |



## Let's understand the read Operation.

Look at the following Two Screen shots (1<sup>st</sup> is Showing Code whereas 2nd is showing different outputs) where comparison among different type of read operation have shown. Here you find comparison among read(),readline() and readlines() functions .

### Screenshot 1

```

fileRead.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\fileRead.py (3.6.5)
File Edit Format Run Options Window Help
print(".....Reading All Characters.....")
f=open("mydat.txt","r")
contents=f.read()
print(contents)
f.close()

print("\n\n.....Reading 70 Characters.....")
f=open("mydat.txt","r")
contents=f.read(70)
print(contents)
f.close()

print("\n\n.....Reading a Single Line.....")
f=open("mydat.txt","r")
contents=f.readline()
print(contents)
f.close()

print("\n\n.....Reading 60 characters from line.....")
f=open("mydat.txt","r")
contents=f.readline(60)
print(contents)
f.close()

print("\n\n.....Reading All Lines.....")
f=open("mydat.txt","r")
contents=f.readlines()
print(contents)
f.close()

```



Showing Code with different types of read operation

### Screenshot 2

```

File Edit Shell Debug Options Window Help
Python is one of those rare languages which is simple and powerful.
You will find it very easy to learn.
The official introduction to Python is:
Python is an easy to learn, powerful programming language.
It has efficient high-level data structures.

.....Reading 70 Characters.....
Python is one of those rare languages which is simple and powerful.
Yo

.....Reading a Single Line.....
Python is one of those rare languages which is simple and powerful.

.....Reading 60 characters from line.....
Python is one of those rare languages which is simple and po






.....Reading All Lines.....
['Python is one of those rare languages which is simple and powerful.\n', 'You will find
it very easy to learn.\n', 'The official introduction to Python is:\n', 'Python is an easy
to learn, powerful programming language.\n', 'It has efficient high-level data structures.
\n']

.....ENJOYED READ OPERATION .....

```



Showing Output of different types of read operations

| CODING   | OUTPUT & EXPLANATION   |
|--|--|
| <pre>f=open("mydat.txt","r") contents=f.read() print(contents) f.close()</pre>       | <pre>.....Reading All Characters..... Python is one of those rare languages which is simple and powerful. You will find it very easy to learn. The official introduction to Python is: Python is an easy to learn, powerful programming language. It has efficient high-level data structures.</pre> <div data-bbox="596 443 1267 555" style="background-color: #c00000; color: white; padding: 5px; border: 1px solid black;">       Look, it read all the characters from the file mydat.txt     </div>                           |
| <pre>f=open("mydat.txt","r") contents=f.read(70) print(contents) f.close()</pre>     | <pre>.....Reading 70 Characters..... Python is one of those rare languages which is simple and powerful. Yo</pre> <div data-bbox="596 786 1311 904" style="background-color: #0070c0; color: white; padding: 5px; border: 1px solid black;">       Here it has read first 70 characters from the file mydat.txt     </div>    |
| <pre>f=open("mydat.txt","r") contents=f.readline() print(contents) f.close()</pre>   | <pre>.....Reading a Single Line..... Python is one of those rare languages which is simple and powerful.</pre> <div data-bbox="596 1122 1283 1211" style="background-color: #c00000; color: white; padding: 5px; border: 1px solid black;">       Here it has read Only first linefrom the file mydat.txt     </div>    |
| <pre>f=open("mydat.txt","r") contents=f.readline(60) print(contents) f.close()</pre> | <pre>.....Reading 60 characters from line..... ..... Python is one of those rare languages which is simple and po</pre> <div data-bbox="596 1429 1267 1541" style="background-color: #0070c0; color: white; padding: 5px; border: 1px solid black;">       Here it has read first 60 characters from the first line of the file mydat.txt     </div>    |
| <pre>f=open("mydat.txt","r") contents=f.readlines() print(contents) f.close()</pre>  | <pre>.....Reading All Lines..... ['Python is one of those rare languages which is simple and powerful.\n', ' You will find it very easy to learn.\n', 'The official introduction to Pyt hon is:\n', 'Python is an easy to learn, powerful programming language.\n', 'It has efficient high-level data structures.\n']</pre> <div data-bbox="596 1832 1294 1944" style="background-color: #c00000; color: white; padding: 5px; border: 1px solid black;">       Here it has read all the lines from the file mydat.txt     </div>  |



## Now let's discuss write Operation on file

Look at the following Two Screen shots (1<sup>st</sup> is Showing Code whereas 2<sup>nd</sup> is showing different outputs) where comparison among different type of write operations have shown. Here you find comparison between write(), writeline()

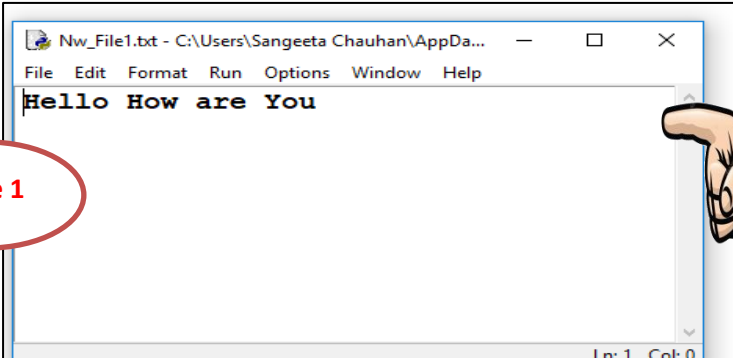
```
fileWrite.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\...
File Edit Format Run Options Window Help
fh = open("Nw_File1.txt", "w")
text = "Hello How are You "
fh.write(text)

fh = open("Nw_File2.txt", "w")
text = ["First line ", "Second line", "Third line "]
fh.writelines(text)
fh.close()

fh = open("Nw_File3.txt", "w")
text = ["First line\n ", "Second line\n", "Third line\n"]
fh.writelines(text)
fh.close()

fh = open("Nw_File4.txt", "w")
text = ("First line\n ", "Second line\n", "Third line \n")
fh.writelines(text)
fh.close()
```

In the above code we have created 4 different files namely **Nw\_File1.txt**, **Nw\_File2.txt**, **Nw\_File3.txt**, **Nw\_File4.txt**.



Case 1

In the Nw\_File1.txt we are writing **String** with **write()** function. See the output

```
Nw_File2.txt - C:\Users\Sangeeta Chauhan\...
File Edit Format Run Options Window Help
First line Second lineThird line
Ln: 1 Col: 0
```

Case 2

In the Nw\_File2.txt we have wrote **list of String without \n** using `writelines()`

```
*Nw_File3.txt - C:\Users\Sangeeta Chauhan\AppData\Local\Pro...
File Edit Format Run Options Window Help
First line
Second line
Third line
```

Case 3

In the Nw\_File3.txt we have wrote **list of String with \n** using `write()` function

```
*Nw_File4.txt - C:\U...
File Edit Format Run Options Window Help
First line
Second line
Third line
```

Case 4

In the Nw\_File4.txt we have wrote **tuple of Strings with \n String** using `write()` function



In the previous cases whenever we run the code again previously written contents will be overwritten. If we want to add contents after the previously added contents then we need to open file in append mode.

See Example “fileappend.py”:

```
python 3.6.2 Shell
fileappend.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\fileappend.py (3.6.5)
File Edit Format Run Options Window Help
f=open('StreamRecord.dat','a') # opening file in append mode
n=int(input('How many records you want to enter'))

for i in range(n):
    print('Record No:', i+1)
    rno=input('Enter Roll no')
    stream=input('Enter Opted Stream')
    record=rno + " " + stream + "\n"
    f.write(record)
f.close()
```

## Output of the above code

```
RESTART: C:/Users/Sangeeta Chauhan/AppData/Local/Programs/Python/Python36-32/fi
leappend.py
How many records you want to enter2
Record No: 1
Enter Roll no 101
Enter Opted StreamScience
Record No: 2
Enter Roll no102
```

## Now reading the contents of file created by above code

```
filereadapp.py - C:/Users/Sangeeta Chauhan/AppData/Local/Programs/Python/Python3
File Edit Format Run Options Window Help
f=open('StreamRecord.dat', 'r')
rec=""
while rec:
    rec=f.readline()
    print(rec)
f.close()
```

File : "filereadapp.py"

Output will be

```
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 (Int
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Sangeeta Chauhan/AppData/Local/Programs/Python/Python36-32:/
lereadapp.py
101 Science

102 Commerce
```

Now If we Run the file "fileappend.py" again it will add new records after the previously added records.

Have a look ,(This time we are adding 3 records)

```
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 (Inte
l)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/Sangeeta Chauhan/AppData/Local/Programs/Python/Python36-32/fi
leappend.py
How many records you want to enter 3
Record No: 1
Enter Roll no 103
Enter Opted StreamCommerce
Record No: 2
Enter Roll no104
Enter Opted StreamCommerce
Record No: 3
Enter Roll no105
Enter Opted StreamScience
>>>
```



After running the "Filereadapp.py" again, Output will be

```
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: C:/Users/Sangeeta Chauhan/AppData/Local/Programs/Python/Python36-32/filereadapp.py
101 Science
102 Commerce
103 Commerce
104 Commerce
105 Science
>>>
```

## 2. Binary Files in Python



Why Binary Files ??????????????

Since binary files store data after converting it into binary language (0s and 1s), there is no EOL character. This file type returns **bytes**. This is the file to be used when **dealing with non-text files** such as **images** or **exe**.



### 2.1 Reading and Writing **list** from/to binary file

Case 1: **Writing list** on Binary File

```
list1 = [23, 43, 55, 10, 90]
newFile = open("binaryFile", "wb")
newFile.write(bytearray(list1))
newFile.close()
```

Conversion of **list** into **bytes** is required

## Case 2: **Reading list** from binary File

```
newFile = open("binaryFile", "rb")
list1=list(newFile.read())
newFile.close()
print(list1)
```

Conversion of **binary data** into **list** is required



## 2.2 Reading and Writing **pdf file** from/to binary file



```
file = open("NewXI1.pdf", "wb") # file to write to
file2 = open("ComputerSc_NewXI.pdf", "rb") # file to read from
bin_cont = file2.read()
file.write(bin_cont)
file.close()
file2.close()
```

Above Code will make another copy of **ComputerSc\_NewXI.pdf** with new name **NewXI1.pdf**



## 2.3 Reading and Writing **image file** from/to binary file

```
file1 = open("TREE.jpg", "rb") # file to read from
file2 = open("Nw_TREE.jpg", "wb") # file to write to
bin_cont = file1.read()
file2.write(bin_cont)
file1.close()
file2.close()
```

Above Code will make another copy of **ComputerSc\_NewXI.pdf** with new name **NewXI1.pdf**

Above Code will make another copy of image file **TREE.jpg** with new name **Nw\_TREE.jpg**



## 2.3 Reading and Writing **Sequences(Dictionary, lists etc) with mixed data type** from/to binary file



To write mixed type of Data we need to **import Pickle module**. **Pickling** means converting structure into byte stream before writing the data into file

Its two main methods are:

1. pickle .dump() : To write the Object into the file

Syntax : `pickle.dump(object_to_write,file_object)`

2. pickle .load() : To read the Object from the file

Syntax : `container_obj=pickle.load(file_object)`

Lets Understand it more clearly through examples

### Case 1: **To Write /Read List**

```
FilePickle1.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\FilePickle1.py ...
File Edit Format Run Options Window Help
import pickle
MyList=[1,2,3,4,5,6,7]
f=open('MyFile.txt','wb') # Opening file in write + binary mode
pickle.dump(MyList,f)
f.close()

f=open('MyFile.txt','rb') # Opening file in read + binary mode
MyList=pickle.load(f)
print('List in File is ',MyList)
f.close()
Ln: 1 Col: 0
```

Output

```
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: C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\P
ython36-32\FilePickle1.py
List in File is [1, 2, 3, 4, 5, 6, 7]
>>>
```

## Case 2: To Write /Read Dictionary

```
FilePickle2.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\FilePickle2.py (3.6.5)
File Edit Format Run Options Window Help

# Writing Dictionary in File
import pickle
MyDict={'list1':[1,2,3], 'list2':[4,5,6], 'list3':[7,8,9]}
f=open('MyFile2.txt','wb') # Opening file in write + binary mode
pickle.dump(MyDict,f)
f.close()

# Reading Dictionary in File
f=open('MyFile2.txt','rb') # Opening file in read + binary mode
MyDict=pickle.load(f)
print('My Dictionary in File is ',MyDict)
f.close()
```

Output

```
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: C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\FilePickle2.py
My Dictionary in File is {'list1': [1, 2, 3], 'list2': [4, 5, 6], 'list3': [7, 8, 9]}
>>>
```

## Case 3: To Write /Read Two different Dictionary

```
FilePickle3.py - C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\FilePickle3.py (3.6.5)
File Edit Format Run Options Window Help

# Writing two Different Dictionaries in File
import pickle
MyDict1={'list1':[1,2,3], 'list2':[4,5,6], 'list3':[7,8,9]}
MyDict2={'Rno':[1,2], 'Name':['Manasvi', 'Divyaditya']}
f=open('MyFile2.txt','wb') # Opening file in write + binary mode
pickle.dump(MyDict1,f) # Writing Object on file
pickle.dump(MyDict2,f)
f.close()

# Reading Dictionary in File
f=open('MyFile2.txt','rb') # Opening file in read + binary mode
MyDict1=pickle.load(f) # Reading first Dictionary
print('My Dictionary 1 in File is ',MyDict)

MyDict2=pickle.load(f) # Reading Second Dictionary
print('My Dictionary 2 in File is ',MyDict2)
f.close()
```

Output

```
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: C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32\FilePickle3.py
My Dictionary 1 in File is {'list1': [1, 2, 3], 'list2': [4, 5, 6], 'list3': [7, 8, 9]}
My Dictionary 2 in File is {'Rno': [1, 2], 'Name': ['Manasvi', 'Divyaditya']}
>>>
```

### 3. ABSOLUTE AND RELATIVE PATHS

- We all know that when we create python files, they are kept in default directory which are also known as folders.
- At the time of Program run Python searches current(default) directory.

**Path :** is the general form of the name of a **file** or directory, specifies a unique location in a **file** system

**Relative Path :** A **relative path** defines a location that is **relative** to the current directory or **folder**. For example, the **relative path** to a **file** named "MyFile.txt" located in the current directory is simply the filename, or "MyFile.txt"

**Absolute Path :** An **absolute path** refers to the complete details needed to locate a **file** or folder, starting from the root element and ending with the other subdirectories

If we want to know the Current Working Directory We have **OS module (Operating System )** which provides many such functions which can be used to work with files and directories and a function **getcwd( )** function can be used to identify the current working directory like this :

```
>>> import os
>>> curr_dir=os.getcwd()
>>> print(curr_dir)
```

**Output** → C:\Users\Sangeeta Chauhan\AppData\Local\Programs\Python\Python36-32

### 4. Standard FileStreams

**standard streams** are preconnected input and output communication channels between a computer program and its environment when it begins execution.

The three input/output (I/O) connections/Streams are :

- Standard input (stdin),** → **sys.stdin** (read from standard input)
- Standard output(stdout)** → **sys.stdout** (Write to Standard Output Device)
- Standard error (stderr).** → **sys.stderr** (Contains Error Messages)