# PYTHON FILE HANDLING

Whatever data stored in python collections like List, Set, and Dict etc. is temporary. Once Python Virtual machine shutdown’s all these objects will be destroyed and data will be gone

Sometimes as the part of programming requirement, we have to store our data permanently for the future purpose. To store data permanently the following are various possibilities

1. Files
2. Databases
3. More advanced storage areas like Big Data

Files are very common permanent storage areas to store our data permanently

TYPES OF FILES

There are two types of files

1. Text files
2. Binary files

**Text files**:

Usually we can use text files to store character data/ text data

**Example:** abc.txt, test.py

**Binary files**:

We can use binary files to store binary data like images, video files, audio files etc**.**

**Example:** guidoVanRossum.jpg, bahubali.mp4

OPENING A FILE

Before performing any operation (like read or write) on the file, first we have to open that file. For this we should use Python's inbuilt function **open ()**

But at the time of open, we have to specify mode, which represents the purpose of opening file.

**f = open (filename, mode)**

The allowed modes in Python are:

1. **r → READ :**
   1. **f = open(‘abc.txt’,’r’)**
   2. Open an existing file for read operation.
   3. The file pointer is positioned at the beginning of the file.
   4. If the specified file does not exist then we will get **FileNotFoundError**.
   5. This is default mode.
2. **w → WRITE :**
   1. **f = open(‘abc.txt’,’w’)**
   2. Open an existing file for write operation.
   3. If the file already contains some data then it will be overwritten.
   4. If the specified file is not already available then this mode will create that file.
3. **a → APPEND :**
   1. **f = open(‘abc.txt’,’a’)**
   2. Open an existing file for append operation.
   3. It won't override existing data.
   4. If the specified file is not already available then this mode will create a new file.
4. **r+ → READ & WRITE :**
   1. **f = open(‘abc.txt’,’r+’)**
   2. To read and write data into the file.
   3. The previous data in the file will not be deleted.
   4. The file pointer is placed at the beginning of the file.
   5. While writing old data will be overwritten
   6. If the specified file does not exist then we will get **FileNotFoundError** and this mode won’t create any new file
5. **w+→ WRITE & READ :**
   1. **f = open(‘abc.txt’,’w+’)**
   2. To write and read data.
   3. It will override existing data.
   4. If the specified file is not already available then this mode will create that file
6. **a+ →** **APPEND & READ:**
   1. **f = open(‘abc.txt’,’a+’)**
   2. To append and read data from the file
   3. It won’t override existing data
   4. If the specified file is not already available then this mode will create that file
7. **x → EXCLUSIVE :**
   1. To open a file in exclusive creation mode for write operation.
   2. If the file already exists then we will get **FileExistsError**.

MODES FOR BINARY FILES

All the above modes are applicable for text files.

If the above modes are suffixed with 'b' then these represents for binary files.

**Example**: rb,wb,ab,r+b,w+b,a+b,xb

**f = open("abc.txt","w")**

We are opening abc.txt file for writing data

**FAQ:**

1. In which modes file should be already there → r, r+
2. In which modes file should not be already there → x
3. In which modes overwriting of existing data will be happened → w, r+, w+
4. In which modes overwriting of existing data won’t be happened → a, a+
5. In which modes new files will be created → w, a, w+, a+, x

CLOSING A FILE

After completing our operations on the file, it is highly recommended to close the file.

For this we have to use close () function

**Syntax = f.close ()**

**Example:**

**f.open(‘abc.txt’,’mode of operation’)**

**# perform required write and read operations on the file**

**f.close()**

Whenever we are closing the file, all system resources which are associated with that file will be released

If we are not closing the file, then there may be performance issues

VARIOUS PROPERTIES OF FILE OBJECT

Once we opened a file and we will get a file object, we can get various details related to that file by using its attributes and methods

**f = open (‘abc.txt’,’w)**

* **f.name →** Name of opened file
* **f.mode →** Mode in which the file is opened
* **f.closed →** Returns Boolean value indicates that file is closed or not
* **f.readable() →** Return’s Boolean value indicates that whether file is readable or not
* **f.writable()** → Returns Boolean value indicates that whether file is writable or not

**Example:**



WRITING DATA TO TEXT FILES:

We can write character data to the text files by using the following 2 methods.

1. **write(str)**
2. **writelines(list of lines)**

**f.write(str)**

**Example:**



**Note:** While writing data by using **write()** methods, compulsory we have to provide the line separator(\n), otherwise total da ta should be written to a single line.

**Example:**



**Note:** In the above program, data present in the file will be overridden every time if we run the program. Instead of overriding if we want append operation then we should open the file as follows.

**f = open("abcd.txt","a")**

**Example:**

****

**f.writelines(list of strings)**

* Instead of list, we can pass tuple, set, and dictionary also
* In the case of sets order will not be guranteed
* But in the case of dictionary, only keys will be added, and keys should be of string type only. Otherwise we will get **TypeError**
* In the case of dictionary, only keys will be added. Instead of keys if we want to add values then we have to write the code explicitly.
* While writing data by using **write(), writelines ()** methods compulsory we have to provide line separator (\n) explicitly, otherwise total data will be written to a single file

**Example:**

****

We can use separator (\n) to separate the strings

**Example:**



**Example:**

****

**Write a program to specify filename and data dynamically from the keyboard?**

****

READING CHARACTER DATA FROM TEXT FILES

We can read character data from text file by using the following read methods.

* **read() →** To read total data from the file
* **read(n) →** To read 'n' characters from the file
* **readline() →** To read only one line
* **readlines() →** To read all lines into a list

**read () :**

**Example:** To read total data from the file



**read (n) :**

* if the specified number of characters are not available in the file then only available characters we will get

**data = f.read (10000)**

* if we are passing negative value as an argument, then we will get total file content

**data = f.read (-1)**

* for any negative value result is always the same

**Example 1:** To read only first 20 characters

****

**Example 2:**

****

**Example 3:**

****

**abc.txt file :**

****

**readlines() :**

**Example 1:** : To read data line by line

****

**Example: 2**

****

**Note:** if the control reaches end of line (i.e. if there is no next line in the file) then readline() method returns an empty string

**readlines() :**

**Example:** To read all lines into a list



**Using multiple read statements together:**

* it is possible to use multiple read statements together, but it is not recommended to use them together

**Example:**

****

**Write a program to read data from one file and write to another file ?**

****

WITH STATEMENT

The With statement can be used while opening a file. We can use this to group file operation statements within a block.

The advantage of with statement is it will take care of closing the file automatically, after completing all operations even in the case of exceptions also, and we are not required to close explicitly

**Example:**

****

**What’s the difference between f=open(‘abc.txt’,’w’) and with open(‘abc.txt’,’w’) as f:**

**f=open(‘abc.txt’,’w’)**

* in this case we have to close the file explicitly

**open(‘abc.txt’,’w’) as f:**

* In this case we are not required to close the file explicitly and it will be closed automatically

THE seek() AND tell() METHODS

**tell():**

* We can use tell() method to return current position of the cursor(file pointer) from beginning of the file. [ can you please tell current cursor position]
* The position (index) of first character in files is zero just like string index.

**Example:**

****

**seek():**

* We can use seek() method to move cursor (file pointer) to specified location. [Can you please seek the cursor to a particular location]

**Python 2.x terminology**

* f.seek(offset, fromWhere) → offset represents the number of positions
* The allowed Values for 2nd Attribute (from where) are:
  + 0 → From beginning of File (Default Value)
  + 1 → From Current Position
  + 2 → From end of the File

**Python 3.x terminology**

* Only zero allowed in python 3
* Hence we can always seek only from beginning of the file only

**f.seek(3)**

**Note:** Python 2 supports all 3 values but python 3 supports only zero

**Example:**

****

**Example:**



HOW TO CHECK A PARTICULAR FILE EXISTS OR NOT ?

We can use OS library to get information about files in our computer.

We can check whether a particular file exists or not by using the isFile() function

OS module has path sub module, which contains **isFile()** function to check whether a particular file exists or not?

**os.path.isfile(fname)**

**Write a Program to check whether the given File exists OR not. If it is available then print its content?**

****

**Write a program to print the Number of Lines, Words and Characters present in the given File?**

****

**Note:** **sys.exit(0)**

* To exit system without executing rest of the program.
* Here argument represents status code.
* 0 means normal termination and it is the default value.

HANDLING BINARY DATA

It is very common requirement to read or write binary data like images, video files, audio files etc.

**Write a program to read Image File and write to a New Image File?**

****

**Note:**

* Even though we can use file handling on binary files like images, videos. It is recommended to use specific libraries like pillow, Open cv for handling binary files

HANDLING CSV FILES

* CSV → Comma Separated Values
* As the part of programming, it is very common requirement to write and read data write csv files
* Python provides csv module to handle csv files.

**Note:** Observe the difference with newline attribute and without

**with open("emp.csv","w",newline='') as f:**

**with open("emp.csv","w") as f:**

**Note:** If we are not using newline attribute then in the csv file blank lines will be included between data. To prevent these blank lines, newline attribute is required in Python-3,but in Python-2 just we can specify mode as 'wb' and we are not required to use newline attribute.

**Write a program to write data to a CSV file?**

****

READING DATA FROM CSV FILE



ZIPPING AND UPZIPING OF FILES

It is very common requirement to zip and unzip files.

The main advantages are:

1. Convenient to use
2. To improve memory utilization
3. We can reduce transport time
4. We can improve performance.

To perform zip and unzip operations, Python contains one in-built module zip file. This module contains a class: ZipFile

**Module name:** zipfile

**Class name:** ZipFile

**How to create a zip file?**

We have to create ZipFile class object with name of the zip file, mode & constant **ZIP\_DEFLATED**. This constant represents we are creating zip file.

**f = ZipFile("files.zip","w","ZIP\_DEFLATED")**

Once we create ZipFile object, we can add files by using **write()** method.

**f.write(filename)**

**Example:**

****

**How to perform unzip Operation**

We have to create ZipFile object as follows **f = ZipFile("files.zip","r",ZIP\_STORED)**

**ZIP\_STORED** constant represents unzip operation.

This is default value and hence we are not required to specify.

Once we created ZipFile class object for unzip operation, we can get all file names present in that zip file by using **namelist()** method

**names=f.namelist()**