# Chapter 3 :

## Computer Science

Class XII ( As per CBSE Board)

New Syllabus 2019-20

Visit : python.mykvs.in for regular updates

# File Handling

A file is a sequence of bytes on the disk/permanent storage where a group of related data is stored. File is created for permanent storage of data.

In programming, Sometimes, it is not enough to only display the data on the console. Those data are to be retrieved later on,then the concept of file handling comes. It is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the file system which is not volatile and can be accessed every time. Here, comes the need of file handling in Python.

File handling in Python enables us to create, update, read, and delete the files stored on the file system through our python program. The following operations can be performed on a file.

In Python, File Handling consists of following three steps:

- Open the file.
- Process file i.e perform read or write operation.
- Close the file.

# File Handling

**Types of File**

**There are two types of files:**

**Text Files-** A file whose contents can be viewed using a text editor is called a text file. A text file is simply a sequence of ASCII or Unicode characters. Python programs, contents written in text editors are some of the example of text files.

**Binary Files-**A binary file stores the data in the same way as as stored in the memory. The .exe files,mp3 file, image files, word documents are some of the examples of binary files.we can't read a binary file using a text editor.

| Text File | Binary File |
|---|---|
| Its Bits represent character. | Its Bits represent a custom data. |
| Less prone to get corrupt as change reflects as soon as made and can be undone. | Can easily get corrupted, corrupt on even single bit change |
| Store only plain text in a file. | Can store different types of data (audio, text,image) in a single file. |
| Widely used file format and can be opened in any text editor. | Developed for an application and can be opened in that application only. |
| Mostly .txt and .rtf are used as extensions to text files. | Can have any application defined extension. |

# File Handling

## Opening and Closing Files-

To perform file operation ,it must be opened first then after reading ,writing, editing operation can be performed. To create any new file then too it must be opened. On opening of any file ,a file relevant structure is created in memory as well as memory space is created to store contents.

Once we are done working with the file, we should close the file. Closing a file releases valuable system resources. In case we forgot to close the file, Python automatically close the file when program ends or file object is no longer referenced in the program. However, if our program is large and we are reading or writing multiple files that can take significant amount of resource on the system. If we keep opening new files carelessly, we could run out of resources. So be a good programmer , close the file as soon as all task are done with it.

## File Handling

### open Function-

Before any reading or writing operation of any file,it must be opened first
of all.Python provide built in function open() for it.On calling of this function creates file object for file operations.
Syntax
file object = open(<file_name>, <access_mode>,< buffering>)
file_name = name of the file ,enclosed in double quotes.
access_mode= Determines the what kind of operations can be performed with file,like read,write etc.
Buffering = for no buffering set it to 0.for line buffering set it to 1.if it is greater than 1 ,then it is buffer size.if it is negative then buffer size is system default.

# File Handling

## File opening modes-

| Sr. No. | Mode & Description |
|---|---|
| 1 | **r** - reading only.Sets file pointer at beginning of the file . This is the default mode. |
| 2 | **rb** – same as r mode but with binary file |
| 3 | **r+** - both reading and writing. The file pointer placed at the beginning of the file. |
| 4 | **rb+** - same as r+ mode but with binary file |
| 5 | **w** - writing only. Overwrites the file if the file exists. If not, creates a new file for writing. |
| 6 | **wb** – same as w mode but with binary file. |
| 7 | **w+** - both writing and reading. Overwrites . If no file exist, creates a new file for R & W. |
| 8 | **wb+** - same as w+ mode but with binary file. |
| 9 | **a** -for appending. Move file pointer at end of the file.Creates new file for writing,if not exist. |
| 10 | **ab** – same as a but with binary file. |
| 11 | **a+** - for both appending and reading. Move file pointer at end. If the file does not exist, it creates a new file for reading and writing. |
| 12 | **ab+** - same as a+ mode but with binary mode. |

# File Handling

## File object attributes –

- **closed:** It returns true if the file is closed and false when the file is open.
- **encoding**: Encoding used for byte string conversion.
- **mode**: Returns file opening mode
- **name**: Returns the name of the file which file object holds.
- **newlines**: Returns “\r”, “\n”, “\r\n”, None or a tuple containing all the newline types seen.

E.g. Program

```
f = open("a.txt", 'a+')
print(f.closed)
print(f.encoding)
print(f.mode)
print(f.newlines)
print(f.name)
```

OUTPUT

```
False
cp1252
a+
None
a.txt
```

# File Handling

## The close() Method

**close()**: Used to close an open file. After using this method,an opened file will be closed and a closed file cannot be read or written any more.

**E.g. program**

```
f = open("a.txt", 'a+')
print(f.closed)
print("Name of the file is",f.name)
f.close()
print(f.closed)
```

**OUTPUT**

```
False
Name of the file is a.txt
True
```

# File Handling

## The write() Method

It writes the contents to the file in the form of string. It does not return value. Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.

## The read() Method

It reads the entire file and returns it contents in the form of a string. Reads at most size bytes or less if end of file occurs.if size not mentioned then read the entire file contents.

## File Handling

write() ,read() Method based program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
text = f.read()
print(text)
f.close()
```

OUTPUT

```
Welcome to python.mykvs.in
Regularly visit python.mykvs.in
```

## File Handling

**readline([size]) method**: Read no of characters from file if size is mentioned till eof.read line till new line character.returns empty string on EOF.

e.g. program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
text = f.readline()
print(text)
text = f.readline()
print(text)
f.close()
```

OUTPUT

```
Welcome to python.mykvs.in

Regularly visit python.mykvs.in
```

# File Handling

**readlines([size]) method**: Read no of lines from file if size is mentioned or all contents if size is not mentioned.

e.g.program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
text = f.readlines(1)

print(text)
f.close()
```

OUTPUT

```
['Welcome to python.mykvs.in\n']
```

NOTE – READ ONLY ONE LINE IN ABOVE PROGRAM.

## File Handling

Iterating over lines in a file
e.g.program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
for text in f.readlines():
    print(text)
f.close()
```

# File Handling

Processing Every Word in a File

e.g.program

```
f = open("a.txt", 'w')
line1 = 'Welcome to python.mykvs.in'
f.write(line1)
line2="\nRegularly visit python.mykvs.in"
f.write(line2)
f.close()

f = open("a.txt", 'r')
for text in f.readlines():
    for word in text.split(  ):
        print(word)
f.close()
```

OUTPUT

```
Welcome
to
python.mykvs.in
Regularly
visit
python.mykvs.in
```

# File Handling

## Append content to a File

```
f = open("a.txt", 'w')
line = 'Welcome to python.mykvs.in\nRegularly visit python.mykvs.in'
f.write(line)
f.close()
```

```
f = open("a.txt", 'a+')
f.write("\nthanks")
f.close()
```

APPEND CODE

```
f = open("a.txt", 'r')
text = f.read()
print(text)
f.close()
```

## OUTPUT

```
Welcome to python.mykvs.in
Regularly visit python.mykvs.in
thanks
```

# File Handling

## Getting & Resetting the Files Position

The ***tell()*** method of python tells us the current position within the file,where as The ***seek(offset[, from])*** method changes the current file position. If ***from*** is 0, the beginning of the file to seek. If it is set to 1, the current position is used . If it is set to 2 then the end of the file would be taken as seek position. The **offset** argument indicates the number of bytes to be moved.

e.g.program

```
f = open("a.txt", 'w')
line = 'Welcome to python.mykvs.in\nRegularly visit python.mykvs.in'
f.write(line)
f.close()

f = open("a.txt", 'rb+')
print(f.tell())
print(f.read(7)) # read seven characters
print(f.tell())
print(f.read())
print(f.tell())
f.seek(9,0) # moves to 9 position from begining
print(f.read(5))
f.seek(4, 1) # moves to 4 position from current location
print(f.read(5))
f.seek(-5, 2) # Go to the 5th byte before the end
print(f.read(5))
f.close()
```

**OUTPUT**

```
0
b'Welcome'
7
b' to
python.mykvs.in\r\n
Regularly visit
python.mykvs.in'
59
b'o pyt'
b'mykvs'
b'vs.in'
```

## File Handling

**Methods of os module**

**1. The rename() method used to rename the file.**

**syntax**

**os.rename(current_file_name, new_file_name)**

2. The **remove()** method to delete file.

**syntax**

os.remove(file_name)

3.The **mkdir()** method of the **os** module to create directories in the current directory.

**syntax**

os.mkdir("newdir")

4.The *chdir()* method to change the current directory.

**syntax**

os.chdir("newdir")

5.The **getcwd()** method displays the current directory.

syntax

os.getcwd()

6. The rmdir() method deletes the directory.

**syntax**

os.rmdir('dirname')

**e.g.program**

```
import os
print(os.getcwd())
os.mkdir("newdir")
os.chdir("newdir")
print(os.getcwd())
```

# File Handling

## Absolute Path vs Relative Path

The absolute path is the full path to some place on your computer. The relative path is the path to some file with respect to your current working directory (PWD). For example:
Absolute path: C:/users/admin/docs/staff.txt
If PWD is C:/users/admin/, then the relative path to staff.txt would be: docs/staff.txt
Note, PWD + relative path = absolute path.
Cool, awesome. Now, if we write some scripts which check if a file exists.
os.chdir("C:/users/admin/docs")
os.path.exists("staff.txt")
This returns TRUE if stuff.txt exists and it works.
Now, instead if we write,
os.path.exists("C:/users/admin/docs/staff.txt")
This will returns TRUE.
If we don't know where the user executing the script from, it is best to compute the absolute path on the user's system using os and __file__.
__file__ is a global variable set on every Python script that returns the relative path to the *.py file that contains it.

**e.g.program**
```
import os
print(os.getcwd())
os.mkdir("newdir1")
os.chdir("newdir1")
print(os.getcwd())
my_absolute_dirpath =
os.path.abspath(os.path.dirname(__file__))
print(my_absolute_dirpath)
```

# File Handling

Standard input, output, and error streams in python

Most  programs make output to "standard out“,input from "standard in“, and error messages go to standard error).standard output is to monitor and standard input is from keyboard.

e.g.program

```
import sys
a = sys.stdin.readline()
sys.stdout.write(a)
a = sys.stdin.read(5)#entered 10 characters.a contains 5 characters.
#The remaining characters are waiting to be read.
sys.stdout.write(a)
b = sys.stdin.read(5)
sys.stdout.write(b)
sys.stderr.write("\ncustom error message")
```