Python for Class XII _ - Part 5

Topics:

File handling: Need for a data file, Types of file: Text files, Binary files and CSV (Comma separated values) files. *(CSV will be discussed separately)*

Text File: Basic operations on a text file: Open (filename – absolute or relative path, mode) / Close a text file, Reading and Manipulation of data from a text file, Appending data into a text file, standard input /output and error streams, relative and absolute paths.

Binary File: Basic operations on a binary file: Open (filename – absolute or relative path, mode) / Close a binary file, Pickle Module – methods load and dump; Read, Write/Create, Search, Append and Update operations in a binary file.

What is a file?

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).

Why do we need a data file?

Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

Operations on a data file

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 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

How to open a file?

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
f = open("test.txt")    # open file in current directory
f = open("C:/Python33/README.txt")  # specifying full path
```

File modes, Text files vs. Binary files

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode.

The default is reading in text mode. In this mode, we get strings when reading from the file.

On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

Python File Modes

Mode	*Description*
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

```
f = open("test.txt")      # equivalent to 'r' or 'rt'
f = open("test.txt",'w')  # write in text mode
f = open("img.bmp",'r+b') # read and write in binary mode
```

How to close a file Using Python?

When we are done with operations to the file, we need to properly close the file.

Closing a file will free up the resources that were tied with the file and is done using Python close() method.

Python has a garbage collector to clean up unreferenced objects but, we must not rely on it to close the file.

```
f = open("test.txt" )

# perform file operations

f.close()
```

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a try...finally block.

```
try:

   f = open("test.txt"')

   # perform file operations

finally:

   f.close()
```

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop.

The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited.

We don't need to explicitly call the close() method. It is done internally.

```
with open("test.txt", 'r') as f:

   # perform file operations
```

How to write to File Using Python?

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode.

We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

```
with open("test.txt", 'w') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
```

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten.

We must include the newline characters ourselves to distinguish different lines.

How to read files in Python?

To read a file in Python, we must open the file in reading mode.

There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

```
>>> f = open("test.txt",'r')
>>> f.read(4)    # read the first 4 data
'This'

>>> f.read(4)    # read the next 4 data
' is '
```

```
>>> f.read()     # read in the rest till end of file
'my first file\nThis file\ncontains three lines\n'


>>> f.read()  # further reading returns empty sting
''
```

We can see that, the read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading.