FILE HANDLING in Python

Introduction to files

- We have so far created programs in Python that accept the input, manipulate it and display the output.
- But that output is available only during execution of the program and input is to be entered through the keyboard.
- This is because the variables used in a program have a lifetime that lasts till the time the program is under execution.

Introduction to files

- Usually, organisations would want to permanently store information about employees, inventory, sales, etc. to avoid repetitive tasks of entering the same data. Hence, data are stored permanently on secondary storage devices for reusability.

- We store Python programs written in script mode with a .py extension. Each program is stored on the secondary device as a file. Likewise, the data entered, and the output can be stored permanently into a file.

A file is a named location on a secondary storage media where data are permanently stored for later access.

Types of files

- Computers store every file as a collection of 0s and 1s i.e., in binary form. Therefore, every file is basically just a series of bytes stored one after the other.
- There are mainly two types of data files:

  1. text file and
  2. binary file
- A text file consists of human readable characters, which can be opened by any text editor.
- Whereas, binary files are made up of non-human readable characters and symbols, which require specific programs to access its contents.

Text file

- A text file can be understood as a sequence of characters consisting of alphabets, numbers and other special symbols.

  Files with extensions like .txt, .py, .csv, etc. are some examples of text files.

- When we open a text file using a text editor (e.g., Notepad), we see several lines of text. However, the file contents are not stored in such a way internally. Rather, they are stored in sequence of bytes consisting of 0s and 1s.

Text file

- In ASCII, UNICODE or any other encoding scheme, the value of each character of the text file is stored as bytes.

- So, while opening a text file, the text editor translates each ASCII value and shows us the equivalent character that is readable by the human being.

  For example,

  the ASCII value 65 (binary equivalent 1000001) will be displayed by a text editor as the letter 'A' since the number 65 in ASCII character set represents 'A'.

Text file

- Each line of a text file is terminated by a special character, called the End of Line (EOL).

  For example, the default EOL character in Python is the newline (\n). However, other characters can be used to indicate EOL.

- When a text editor or a program interpreter encounters the ASCII equivalent of the EOL character, it displays the remaining file contents starting from a new line.

- Contents in a text file are usually separated by whitespace, but comma (,) and tab (\t) are also commonly used to separate values in a text file.

Binary file

- Like text files, Binary files are also stored in terms of bytes (0s and 1s), but unlike text files, these bytes do not represent the ASCII values of characters. Rather, they represent the actual content such as image, audio, video, compressed versions of other files, executable files, etc.

- These files are not human readable. Thus, trying to open a binary file using a text editor will show some garbage values. We need specific software to read or write the contents of a binary file.

Binary file

- Binary files are stored in a computer in a sequence of bytes. Even a single bit change can corrupt the file and make it unreadable to the supporting application.

- Also, it is difficult to remove any error which may occur in the binary file as the stored contents are not human readable.

Note:

We can read and write both text and binary files through Python programs.

opening and closing a text file

- In real world applications, computer programs deal with data coming from different sources like databases, CSV files, HTML, XML, JSON, etc.

- We broadly access files either to write or read data from it. But operations on files include creating and opening a file, writing data in a file, traversing a file, reading data from a file and so on.

- Python has the io module that contains different functions for handling files.

opening a file

- To open a file in Python, we use the open() function. The syntax of open() is as follows:

```
file_object= open(file_name, access_mode)
```

This function returns a file object called file handle which is stored in the variable file_object.

We can use this variable to transfer data to and from the file (read and write) by calling the functions defined in the Python's io module.

Note:

If the file does not exist, the above statement creates a new empty file and assigns it the name we specify in the statement.

opening a file: file_object

- To open a file in Python, we use the open() function. The syntax of open() is as follows:

  file_object= open(file_name, access_mode)

- The file_object has certain attributes that tells us basic information about the file, such as:

  <file.closed> returns true if the file is closed and false otherwise.

  <file.mode> returns the access mode in which the file was opened.

  <file.name> returns the name of the file.

opening a file: file_name

- To open a file in Python, we use the open() function. The syntax of open() is as follows:

```
file_object= open(file_name, access_mode)
```

- The file_name should be the name of the file that has to be opened. If the file is not in the current working directory, then we need to specify the complete path of the file along with its name.

opening a file: access_mode

- To open a file in Python, we use the open() function. The syntax of open() is as follows:

```
file_object= open(file_name, access_mode)
```

- The access_mode is an optional argument that represents the mode in which the file has to be accessed by the program. It is also referred to as processing mode.
- Here mode means the operation for which the file has to be opened like <r> for reading, <w> for writing, <+> for both reading and writing, <a> for appending at the end of an existing file. The default is the read mode.

opening a file: access_mode

- In addition, we can specify whether the file will be handled as binary (<b>) or text mode.

  By default, files are opened in text mode that means strings can be read or written.

  Files containing non-textual data are opened in binary mode that means read/write are performed in terms of bytes.

Note:

When a file is opened in a particular mode, the file offset position refers to the position of the file object

File Mode	Description	File Offset position
<r>	Opens the file in read-only mode.	Beginning of the file
<rb>	Opens the file in binary and read-only mode.	Beginning of the file
<r+> or <+r>	Opens the file in both read and write mode.	Beginning of the file
<w>	Opens the file in write mode. If the file already exists, all the contents will be overwritten. If the file doesn't exist, then a new file will be created.	Beginning of the file
<wb+> or <+wb>	Opens the file in read,write and binary mode. If the file already exists, the contents will be overwritten. If the file doesn't exist, then a new file will be created.	Beginning of the file
<a>	Opens the file in append mode. If the file doesn't exist, then a new file will be created.	End of the file
<a+> or <+a>	Opens the file in append and read mode. If the file doesn't exist, then it will create a new file.	End of the file

In addition to the above file access modes, we have some other file access modes: <rb+>, <wb>, <w+>, <ab>, <ab+>

opening a file: access_mode

- Consider the following example:

  myObject=open("myfile.txt", "a+")

- In the above statement, the file *myfile.txt* is opened in append and read modes. The file object will be at the end of the file.

- That means we can write data at the end of the file and at the same time we can also read data from the file using the file object named *myObject*.

closing a file

- Once we are done with the read/write operations on a file, it is a good practice to close the file. Python provides a close() method to do so.

- While closing a file, the system frees the memory allocated to it. The syntax of close() is:

```
file_object.close()
```

Here, file_object is the object that was returned while opening the file.

closing a file

- Python makes sure that any unwritten or unsaved data is flushed off (written) to the file before it is closed. Hence, it is always advised to close the file once our work is done.

Note:

If the file object is re-assigned to some other file, the previous file is automatically closed.

Summary

- Introduction to files
- Types of files:
  - Text file
  - Binary file
- Opening and closing a file