File Handling in Python

Before we can go into how to work with files in Python, it's important to understand what exactly a file is and how modern operating systems handle some of their aspects.

What Is a File?

A **file** is a contiguous set of bytes used to store data. This data is organized in a specific format and can be anything as simple as a text file or as complicated as a program executable.

File Paths:

When you access a file on an operating system, a file path is required. The file path is a string that represents the location of a file. It's broken up into three major parts:

Folder Path: the file folder location on the file system where subsequent folders are separated by a forward slash / (Unix) or backslash \ (Windows). *e.g. C:\Vaibhav\Programs\xyz.txt*

File Name: the actual name of the file. *e.g. xyz.txt*

Extension: the end of the file path pre-pended with a period (.) used to indicate the file type *e.g. .txt, .py, .jpg, .mp3, .pdf etc.*

Operating/Processing a File in Python:

Usually we've to follow the following sequence while processing files in Python.

1. Open the file.
2. Perform the operations (Reading, Writing etc.).
3. Close the file.

To perform these operations we'll use many built-in functions like open(), read(), write(), close() and many more….

Opening a File:

When you want to work with a file, the first thing to do is to open it. This is done by invoking the open(), a built-in function.
open() has a single required argument that is the path to the file.
open() returns, the file object, usually called as "handle".
Note: Always make sure that an open file is closed properly.

Syntax: file_handle = open("file_name.extension")
Example: f = open("xyz.txt")

File Opening Modes:

Most likely, you'll also want to use the second positional argument, mode. This argument is a string that contains multiple characters to represent how you want to open the file. The default and most common is 'r'.

Modes	Description	Error
"r"	Open a file for read only. DEFAULT MODE.	If file doesn't exists, generates *FileNotFoundError.*
"w"	Open a file for writing.	-
"a"	Opens a file in append mode.	-
"x"	Create a file.	If file already exists, generates *FileExistsError.*

Additionally, we can specify the type of file as well, such as **"t"** for text files and **"b"** for binary files.
e.g. "rb", "wt" etc.
We'll cover this in live session.

Types of files in Python:

1)Binary Files:
Files having specific encoding, Binary files will be encoded in the binary format, which can be understood only by a computer or machine.
For handling such binary files we need a specific type of software to open it.
Most of the files that we see in our computer system are called binary files.

Examples-

Document files: .pdf, .doc, .xls etc.
Image files: .png, .jpg, .gif, .bmp etc.
Video files: .mp4, .3gp, .mkv, .avi etc.
Audio files: .mp3, .wav, .mka, .aac etc.
Database files: .mdb, .accde, .frm, .sqlite etc.
Archive files: .zip, .rar, .iso, .7z etc.
Executable files: .exe, .dll, .class etc.

2)Text Files:
Text files don't have any specific encoding and it can be opened in normal text editor itself.

Examples-

Web standards: html, XML, CSS, JSON etc.
Source code: c, app, js, py, java etc.
Documents: txt, tex, RTF etc.
Tabular data: csv, tsv etc.
Configuration: ini, cfg, reg etc.

Built-in functions:

Function	Description
open()	The open() function is used to open file.
read()	Reads file and return one big string.
read(n)	Reads 'n' characters from the file till end of the file
readline()	Reads file and return one line at a time.
readlines()	Reads file and returns a list of lines.
write()	Writes a data to a file.
writelines()	Writes a list of strings.
close()	Closes file & free up system resources.
readable()	Returns true if the file is readable
writable()	Returns true if the file is writable
seekable()	Returns true if the file supports random access
seek(offset)	Change the cursor position by bytes as specified
tell()	Returns the current file location

Another approaches used to Close file:

It's important to remember that it's your responsibility to close the file. If not closed properly can lead to unwanted behavior including resource leaks.

When you're manipulating a file, there are two ways that you can use to ensure that a file is closed properly, even when encountering an error.

1)The first way to close a file is to use the try-finally block:

```
f = open('abc.txt')
try:
    # Further file processing goes here
finally:
    f.close()
```

2)The second way to close a file is to use the with statement:

```
with open('abc.txt') as f:
    # Further file processing goes here
```

The with statement automatically takes care of closing the file once it leaves the with block, even in cases of error.