File Handling

- Python supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.
- The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short.
- Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file.
- Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character.
- It ends the current line and tells the interpreter a new one has begun. Let's start with the reading and writing files.

❖ Advantages of File Handling

➢ **Versatility**: File handling in Python allows you to perform a wide range of operations, such as creating, reading, writing, appending, renaming, and deleting files.

➢ **Flexibility**: File handling in Python is highly flexible, as it allows you to work with different file types (e.g. text files, binary files, CSV files, etc.), and to perform different operations on files (e.g. read, write, append, etc.).

➢ **User–friendly**: Python provides a user-friendly interface for file handling, making it easy to create, read, and manipulate files.

➢ **Cross-platform**: Python file-handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.

❖ Disadvantages of File Handling

- **Error-prone:** File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.).
- **Security risks**: File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system.
- **Complexity**: File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely.
- **Performance**: File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.

❖ Working of open() Function in Python

Before performing any operation on the file like reading or writing, first, we have to open that file.

For this, we should use Python's inbuilt function open() but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

Syntax:

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

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

SN	Access mode	Description
1	r	It opens the file to read-only mode. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2	rb	It opens the file to read-only in binary format. The file pointer exists at the beginning of the file.
3	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5	w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
6	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists. The file pointer exists at the beginning of the file.
7	w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.
8	wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.
9	a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.

10	ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.
11	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

❖ Working in Read mode

There is more than one way to read a file in Python.

Example:

```
# Python code to illustrate read() mode character wise

file = open("topper.txt", "r")

print (file.read(5))
```

❖ Creating a File using the write() Function

Just like reading a file in Python, there are a number of ways to write in a file in Python.

Let us see how we can write the content of a file using the write() function in Python.

Example:

```
# Python code to create a file

file = open('Topper.txt','w')

file.write("This is the write command")

file.write("It allows us to write in a particular file")

file.close()
```

Output:

```
This is the write command
It allows us to write in a particular file
```

❖ The close() method

- Once all the operations are done on the file, we must close it through our Python script using the **close()** method.
- Any unwritten information gets destroyed once the **close()** method is called on a file object.
- We can perform any operation on the file externally using the file system which is the currently opened in Python; hence it is good practice to close the file once all the operations are done.

Syntax:

```
fileobject.close()
```

Example:

```
# opens the file file.txt in read mode
fileptr = open("file.txt","r")

if fileptr:
   print("file is opened successfully")

#closes the opened file
fileptr.close()
```

❖ The with statement

- The **with** statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files.
- It is used in the scenario where a pair of statements is to be executed with a block of code in between.

Syntax:

```
with open(<file name>, <access mode>) as <file-pointer>:
    #statement suite
```

- The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.
- It is always suggestible to use the **with** statement in the case of files because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file, we don't need to write the **close()** function. It doesn't let the file to corrupt.

Example:

```
with open("file.txt",'r') as f:
 content = f.read();
print(content)
```

❖ The file related methods

The file object provides the following methods to manipulate the files on various operating systems.

SN	Method	Description
1	file.close()	It closes the opened file. The file once closed, it can't be read or write anymore.

2	File.fush()	It flushes the internal buffer.
3	File.fileno()	It returns the file descriptor used by the underlying implementation to request I/O from the OS.
4	File.isatty()	It returns true if the file is connected to a TTY device, otherwise returns false.
5	File.next()	It returns the next line from the file.
6	File.read([size])	It reads the file for the specified size.
7	File.readline([size])	It reads one line from the file and places the file pointer to the beginning of the new line.
8	File.readlines([sizehint])	It returns a list containing all the lines of the file. It reads the file until the EOF occurs using readline() function.
9	File.seek(offset[,from)	It modifies the position of the file pointer to a specified offset with the specified reference.
10	File.tell()	It returns the current position of the file pointer within the file.
11	File.truncate([size])	It truncates the file to the optional specified size.
12	File.write(str)	It writes the specified string to a file
13	File.writelines(seq)	It writes a sequence of the strings to a file.