**File Handling in Python**

File handling in Python involves interacting with files on your computer to read data from them or write data to them. Python provides several built-in functions and methods for creating, opening, reading, writing, and closing files.

**Opening a File :-**

To perform any file operation, the first step is to open the file. Python's built-in open() function is used to open files in various modes, such as reading, writing, and appending.

The syntax for opening file= open("filename", "mode")

|  |
| --- |
| **Modes:-** |
| 1 | **r**  Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode. |
| 2 | **rb**  Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode. |
| 3 | **r+**  Opens a file for both reading and writing. The file pointer placed at the beginning of the file. |
| 4 | **rb+**  Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file. |
| 5 | **w**  Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| 6 | **b**  Opens the file in binary mode |
| 7 | **t**  Opens the file in text mode (default) |
| 8 | **+**  open file for updating (reading and writing) |
| 9 | **wb**  Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| 10 | **w+**  Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| 11 | **wb+**  Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| 12 | **a**  Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| 13 | **ab**  Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| 14 | **a+**  Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |
| 15 | **ab+**  Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |
| 16 | **x**  open for exclusive creation, failing if the file already exists |
| Examples:- |  |

#Opening a file in read mode

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

# Opening a file in write mode

file = open("example.txt", "w")

# Opening a file in append mode

file = open("example.txt", "a")

# Opening a file in binary read mode

file = open("example.txt", "rb")

Reading a File :-

Reading a file involves opening the file in a mode that allows for reading, and then using various methods to extract the data from the file.

Ex:- with open('example.txt', 'r') as file:  
 content = file.read()  
 print(content)

**Writing to a File :-**

Writing to a file involves opening the file in a mode that allows writing, When opening a file in write mode ('w'), the file's existing content is erased.

Ex:- with open('example.txt', 'w') as file:  
 file.write("Hello, World!")  
 file.write("This is a new line.")

**Closing a File :-**

We can close a file in Python using the close() method. Closing a file is an essential step in file handling to ensure that all resources used by the file are properly released. It is important to close files after operations are completed to prevent data loss and free up system resources.

file = open("example.txt", "w")

file.write("This is an example.")

file.close()

print ("File closed successfully!!")

**Advantages:-**

* **Versatility**: File handling in Python allows you to perform a wide range of operations.
* **Flexibility**: File handling in Python is highly flexible, as it allows you to work with different file types.
* **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:**

* **Error-prone:**File handling operations in Python can be prone to errors.
* **Security risks**: File handling in Python can also pose security risks.
* **Complexity**: File handling in Python can be complex, especially when working with more advanced file formats or operations.
* **Performance**: File handling operations in Python can be slower than other programming

languages.