**Python File Handling**

Python too 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](https://www.geeksforgeeks.org/python-programming-language/) 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()](https://www.geeksforgeeks.org/python-open-function/) but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

f = open(filename, mode)

Where the following mode is supported:

1. **r:**open an existing file for a read operation.
2. **w:** open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.
3. **a:**open an existing file for append operation. It won’t override existing data.
4. **r+:**  To read and write data into the file. The previous data in the file will be overridden.
5. **w+:** To write and read data. It will override existing data.
6. **a+:** To append and read data from the file. It won’t override existing data.

**Working in Read mode**

There is more than one way to [read a file in Python](https://www.geeksforgeeks.org/how-to-read-from-a-file-in-python/). Let us see how we can read the content of a file in read mode.

**Example 1:** The open command will open the file in the read mode and the for loop will print each line present in the file.

* Python3

file = open('geek.txt', 'r')

|  |
| --- |
| file = open('geek.txt', 'r')  for each in file:      print (each) |

**Output:**

Hello world

GeeksforGeeks

123 456

**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](https://www.geeksforgeeks.org/writing-to-file-in-python/). Let us see how we can write the content of a file using the write() function in Python.

**Working in Write Mode**

Let’s see how to create a file and how the write mode works.

**Example 1:** In this example, we will see how the write mode and the write() function is used to write in a file. The close() command terminates all the resources in use and frees the system of this particular program.

* Python3

|  |
| --- |
| file = open('geek.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 commandIt allows us to write in a particular file

## Python Directory

Python contains several modules that has a number of built-in functions to manipulate and process data. Python has also provided modules that help us to interact with the operating system and the files. These kinds of modules can be used for directory management also. The modules that provide the functionalities are listed below:

* os and os.path
* filecmp
* tempfile
* shutil

### os and os.path module

The os module is used to handle files and directories in various ways. It provides provisions to create/rename/delete directories. This allows even to know the current working directory and change it to another. It also allows one to copy files from one directory to another. The major methods used for directory management is explained below.

#### Creating new directory:

* **os.mkdir(name)** method to create a new directory.
* The desired name for the new directory is passed as the parameter.
* By default it creates the new directory in the current working directory.
* If the new directory has to be created somewhere else then that path has to be specified and the path should contain forward slashes instead of backward ones.

#### Getting Current Working Directory (CWD):

* **os.getcwd()**can be used.
* It returns a string that represents the path of the current working directory.
* **os.getcwdb()**can also be used but it returns a byte string that represents the current working directory.
* Both methods donot require any parameters to be passed

#### Renaming a directory:

* **os.rename()**method is used to rename the directory.
* The parameters passed are old\_name followed by new\_name.
* If a directory already exists with the new\_name passed, OSError will be raised in case of both Unix and Windows.
* If a file already exists with the new\_name, in Unix no error arises, the directory will be renamed. But in Windows the renaming won’t happen and error will be raised.
* **os.renames(‘old\_name’,’dest\_dir:/new\_name’)** method works similar to **os.rename()** but it moves the renamed file to the specified destination directory(dest\_dir).

#### Listing the files in a directory

* A directory may contain sub-directories and a number of files in it. To list them, **os.listdir()**method is used.
* It either takes no parameter or one parameter.
* If no parameter is passed, then the files and sub-directories of the CWD is listed.
* If files of any other directory other than the CWD are required to be listed, then that directory’s name/path is passed as parameter.

|  |
| --- |
|  |