File Handling

- **Python has several functions for creating, reading, updating, and deleting files.a**
- Before performing any operation on the file like reading or writing, first, **we have to open that file.**

Methods of File Handling

Assume we have the following file, located in the same folder as Python:
demofile.txt
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!

File Methods

The various file handling methods and their description with example are enlisted in the Table below:

S. No.	Method Name	Description	Example
1.	read()	Returns the file content	f = open("demofile.txt", "r") print(f.read())
2.	readable()	Returns True if the file is readable, False if not	f = open("demofile.txt", "r") print(f.readable())
3.	readline()	Returns one line from the file	f = open("demofile.txt", "r") print(f.readline())
4.	write()	Writes the specified string to the file	f = open("demofile2.txt", "a") f.write("See you soon!") f.close()
5.	writable()	Returns True if the file is writable, False if not.	f = open("demofile.txt", "a") print(f.writable())
6.	writelines()	Writes a list of strings to the file	f = open("demofile3.txt", "a") f.writelines(["See you soon!", "Over and out."]) f.close()
7.	truncate()	Resizes the file to a specified size	f = open("demofile2.txt", "a") f.truncate(20) f.close()
8.	close()	Closes the file	f = open("demofile.txt", "r") print(f.read()) f.close()
9.	fileno()	Returns a number that represents the file descriptor, from the operating system's perspective	f = open("demofile.txt", "r") print(f.fileno())

File Open

- To open the file, the built-in open() function is used.
- While opening a file, we have to specify the mode, which represents the purpose of the opening file.
- The open() function takes two parameters; *filename*, and *mode*.

f = open(filename, mode)

- **Modes**

  There are different methods (modes) for opening a file:
  - **"r"** - Read - Default value. Opens a file for reading, error if the file does not exist
  - **"a"** - Append - Opens a file for appending, creates the file if it does not exist
  - **"w"** - Write - Opens a file for writing, creates the file if it does not exist
  - **"x"** - Create - Creates the specified file, returns an error if the file exists
  - **"r+"** - To read and write data into the file. The previous data in the file will be overridden.
  - **"w+"** - To write and read data. It will override existing data.
  - **"a+"** - To append and read data from the file. It won't override existing data.

- In addition, the user can specify if the file should be handled as binary or text mode:
  - **"t"** - Text - Default value. Text mode
  - **"b"** - Binary - Binary mode (e.g. images)

Syntax:

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
print(f.read())
```

Output:

```
C:\Users\Swati Dewangan>python file_operations.py
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

Note:

- If the file is located in a different location, we will have to specify the file path, like this:

Example:

```
#Open a file on a different location:
f = open("D:\Folder_name\demofile.txt", "r")
print(f.read())
```

Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Example

```
#Return the 5 first characters of the file:
f = open("demofile.txt", "r")
print(f.read(5))
```

Read Lines

- You can return one line by using the readline() method:

Example

```
#Read one line of the file:
f = open("demofile.txt", "r")
print(f.readline())
```

- By calling readline() two times, you can read the two first lines:

Example

```
#Read two lines of the file:
f = open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

- By looping through the lines of the file, you can read the whole file, line by line:

Example

```
#Loop through the file line by line:
f = open("demofile.txt", "r")
for x in f:
  print(x)
```

Close Files

- It is a good practice to always close the file when after use.

Example

```
#To close the file when you are finish working in it:
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

File Write

- To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Examples:

```
#Open the file "demofile2.txt" and append content to the file:
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()
#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
#Open the file "demofile3.txt" and overwrite the content:
f = open("demofile3.txt", "w")
f.write("Sorry! I have deleted the content!")
f.close()
#open and read the file after the appending:
f = open("demofile3.txt", "r")
print(f.read())
```

Create a New File

- To create a new file in Python, use the open() method, with one of the following parameters:

  "x" - Create - will create a file, returns an error if the file exist

  "a" - Append - will create a file if the specified file does not exist

  "w" - Write - will create a file if the specified file does not exist

Example

```
#Create a file called "myfile.txt":
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example

```
#Create a new file if it does not exist:
f = open("myfile.txt", "w")
```

File Delete

To delete a file, you must import the OS module, and run its os.remove() function:

Example

```
#Remove the file "demofile.txt":
```

```
import os
os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example

```
#Check if file exists, then delete it:
import os
if os.path.exists("demofile.txt"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the os.rmdir() method:

Example

```
#Remove the folder "myfolder":
import os
os.rmdir("myfolder")
```

Note: User can only remove *empty* folders.

Directory Handling

- Directories are **a way of storing, organizing, and separating the files on a computer.**
- The **directory that does not have a parent** is called a **root directory.**
- The **way to reach the file** is called the **path**.
- The **path contains a combination of directory names, folder names separated by slashes and colon** and this gives the route to a file in the system.
- Python provides os modules to interact with the directory in any system.

os Module

- The os module is **used to handle files and directories**.
- **It provides provisions to create/rename/delete directories**.
- It also allows one to copy files from one directory to another.
- It allows to know the current working directory and change it to another.

Directory Methods

The various directory handling methods and their description with example are enlisted in the Table below:

S. No.	Methods	Description	Syntax	Example
1.	mkdir()	To create directory	os.mkdir(path)	# Create a directory "test" import os os.mkdir("test")
2.	chdir()	To change the current directory.	os.chdir("newdir path")	# Changing a directory to "/home/newdir" import os os.chdir("/home/newdir")
3.	getcwd()	Displays the current working directory.	os.getcwd()	#To find the location of the current directory import os os.getcwd()
4.	rmdir()	deletes the directory (Before removing a directory, all the contents in it should be removed.)	os.rmdir('dirname')	# To delete or remove "/tmp/test" directory import os os.rmdir("/tmp/test")
5.	rename()	to rename the directory	os.rename('old_name','new_name')	import os os.rename('file1.txt','file.txt')
6.	**listdir()**	Listing the files inside a directory	os.listdir()	import os print(os.listdir('dir path'))
7.	isdir()	To check whether it is a directory	os.path.isdir(path)	import os print(os.path.isdir('dir path'))
8.	getsize()	To get size of the directory in bytes	os.path.getsize(path_name)	import os print(os.path.getsize('dir path'))