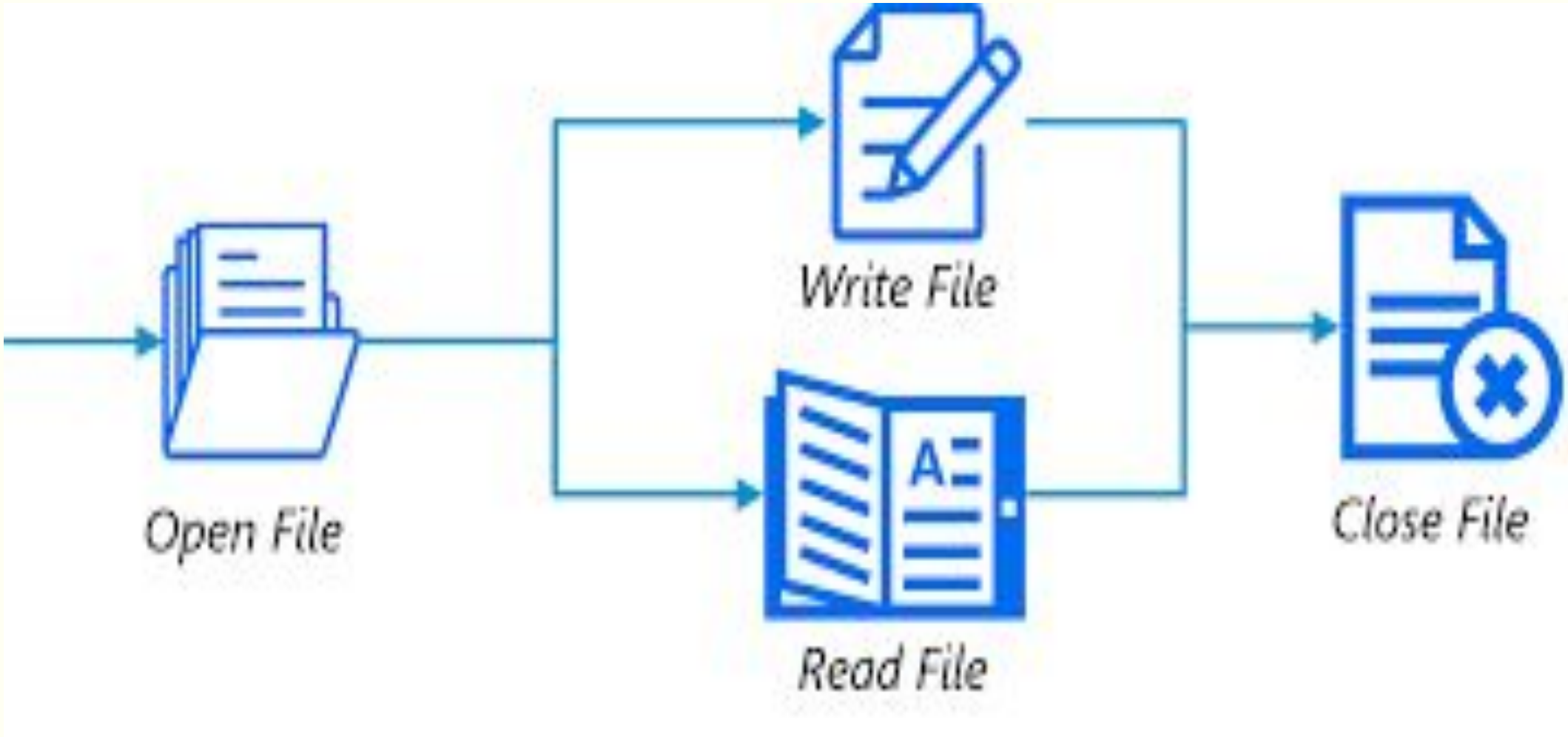
FILE HANDLING

What is File ???

- File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).
- To store data temporarily and permanently, we use files. A file is the collection of data stored on a disk in one unit identified by filename.
- Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.
- When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.
- Hence, in Python, a file operation takes place in the following order.
  - Open a file
  - Read or write (perform operation)
  - Close the file

Basics of File Handling



Modes of Files

- There are different modes of file in which it can be opened. They are mentioned in the following table.
- A File can be opened in two modes:

1) Text Mode.

2) Binary Mode.

File Path

A file path defines the location of a file or folder in the computer system.

There are two ways to specify a file path.

1. **Absolute path**: which always begins with the root folder
2. **Relative path**: which is relative to the program's current working directory

The absolute path includes the complete directory list required to locate the file.

For example, /user/Pynative/data/sales.txt is an absolute path to discover the sales.txt.

All of the information needed to find the file is contained in the path string.

After the filename, the part with a period(.) is called the file's extension, and that tells us the type of file.
Here, project.pdf is a pdf document.

Open a File

1) **Opening a File:** Before working with Files you have to open the File. To open a File, Python built in function open() is used. It returns an object of File which is used with other functions. Having opened the file now you can perform read, write, etc. operations on the File.

Syntax:

fileobj=open(filename , mode)

- **filename:**It is the name of the file which you want to access.
- **mode:**It specifies the mode in which File is to be opened. There are many types of mode. Mode depends the operation to be performed on File. Default access mode is read.

Create a File

We don't have to import any module to create a new file. We can create a file using the built-in function `open()`.

```
open('file_Path', 'access_mode')
```

File Mode	Meaning
`w`	Create a new file for writing. If a file already exists, it truncates the file first. Use to create and write content into a new file.
`x`	Open a file only for exclusive creation. If the file already exists, this operation fails.
`a`	Open a file in the append mode and add new content at the end of the file.
`b`	Create a binary file
`t`	Create and open a file in a text mode

Opening a File with Relative Path

- A relative path is a path that starts with the working directory or the current directory and then will start looking for the file from that directory to the file name.

```
try:
    fp = open(r'E:\PYnative\reports\samples.txt', 'r')
    print(fp.read())
    fp.close()
except IOError:
    print("File not found. Please check the path.")
finally:
    print("Exit")
```

File open() function

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None,
```

```
errors=None, newline=None, closefd=True, opener=None)
```

Parameter	Description
file	This parameter value gives the pathname (absolute or relative to the current working directory) of the file to be opened.
mode	This is the optional string that specifies the mode in which a file will be opened. The default value is 'r' for reading a text file. We can discuss the other modes in the later section.
buffering	This is an optional integer used to set the buffering policy. Pass 0 to switch buffering off (only allowed in binary mode), 1 to select line buffering (only usable in text mode), and an integer > 1 to indicate the size in bytes of a fixed-size chunk buffer.
encoding	This is the name of the encoding used to decode or encode the file. The default one is platform dependant.
errors	These are optional string denotes how the standard encoding and decoding errors have to be handled.
newline	This is the parameter that indicates how the newline mode works (it only applies to text mode). It can be None, '', '\n', '\r', and '\r\n'.
closefd	This parameter indicates whether to close a file descriptor or not. The default value is True. If closefd is False and a file descriptor rather than a filename was given, the underlying file descriptor will be kept open when the file is closed.

Example: **Create a new empty text file** named 'sales.txt'

```
# create a empty text file
# in current directory
fp = open('sales.txt', 'x')
fp.close()
```

Use **access mode** `w` if you want to create and write content into a file.

```
# create a empty text file
fp = open('sales_2.txt', 'w')
fp.write('first line')
fp.close()
```

Close a File

2) Closing a File: Once you are finished with the operations on File at the end you need to close the file. It is done by the close() method. close() method is used to close a File.

Syntax:

fileobject.close()

Modes of Files

Mode	Description
R	It opens in Reading mode. It is default mode of File. Pointer is at beginning of the file.
rb	It opens in Reading mode for binary format. It is the default mode. Pointer is at beginning of file.
r+	Opens file for reading and writing. Pointer is at beginning of file.
rb+	Opens file for reading and writing in binary format. Pointer is at beginning of file.
W	Opens file in Writing mode. If file already exists, then overwrite the file else create a new file.
wb	Opens file in Writing mode in binary format. If file already exists, then overwrite the file else create a new file.
w+	Opens file for reading and writing. If file already exists, then overwrite the file else create a new file.
wb+	Opens file for reading and writing in binary format. If file already exists, then overwrite the file else create a new file.
a	Opens file in Appending mode. If file already exists, then append the data at the end of existing file, else create a new file.
ab	Opens file in Appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.
a+	Opens file in reading and appending mode. If file already exists, then append the data at the end of existing file, else create a new file.
ab+	Opens file in reading and appending mode in binary format. If file already exists, then append the data at the end of existing file, else create a new file.

File Attributes :Metadata about working file

- Once a file is opened and you have one *file* object, you can get various information related to that file.
- Here is a list of all attributes related to file object –

Attribute	Description
Name	Returns the name of the file.
Mode	Returns the mode in which file is being opened.
Closed	Returns Boolean value. True, in case if file is closed else false.

Write a File

3) **Writing to a File:**write() method is used to write a string into a file.

- **Syntax:**
  - Fileobject.write(string str)

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

file.write("Hello World")

file.write("This is our new text file")

file.write("and this is another line.")

file.write("Why? Because we can.")

file.close()

- write multiple lines to a file at once

```
>>> fh = open("hello.txt","w")
>>> lines_of_text = ["One line of text here", "and another line
  here", "and yet another here", "and so on and so forth"]
>>> fh.writelines(lines_of_text)
>>> fh.close()
```

- To append a file

```
>>> fh = open("hello.txt", "a")
>>> fh.write("We Meet Again World")
>>> fh.close
```

Read a File

- **Reading from a File:**read() method is used to read data from the File.
- **Syntax:**
  - fileobject.read(value)
- here, value is the number of bytes to be read. In case, no value is given it reads till end of file is reached.

Variations in Read Mode

1) read() or read(4)

Use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

Eg. ***file = open("testfile.text", "r")***

```
print file.read()        #till end of file
print file.read(5)       #only first 5 bytes
```

2)readline() or readline(3)

Use readline() method to read individual lines of a file. This method reads only the first line of the file.

3)readlines()

reads all the lines and put them in 'list'(between these [ ]) and displays them

Looping over a file object

- read – or return – all the lines from a file in a more memory efficient, and fast manner

Eg. file = open("testfile.txt", "r")
for line in file:
*** print line***

- use the with statement to open a file

Eg. ***with open("testfile.txt") as f:***
for line in f:
*** print line***

- Splitting Lines in a Text File

Eg. ***with open("hello.text", "r") as f:***
data = f.readlines()
for line in data:
*** words = line.split()***
print words

File Methods

Method	Description
close()	Close an open file. It has no effect if the file is already closed.
detach()	Separate the underlying binary buffer from the `TextIOBase`and return it.
fileno()	Return an integer number (file descriptor) of the file.
flush()	Flush the write buffer of the file stream.
isatty()	Return `True` if the file stream is interactive.

File Methods

Method	Description
read(n)	Read atmost n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Read and return one line from the file. Reads in at most nbytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.

File Methods

Method	Description
tell()	Returns the current file location.
truncate(`size=None`)	Resize the file stream to `size` bytes. If `size` is not specified, resize to current location.
writable()	Returns `True` if the file stream can be written to.
write(`s`)	Write string `s` to the file and return the number of characters written.
writelines(`lines`)	Write a list of `lines` to the file.

I/O Module

- The io module provides Python's main facilities for dealing for various types of I/O.
- There are three main types of I/O: *text I/O*, *binary I/O*, *raw I/O*.
- These are generic categories, and various backing stores can be used for each of them.
- Concrete objects belonging to any of these categories will often be called *streams*;
- Another common term is *file-like objects*. Independently of its category, each concrete stream object will also have various capabilities: it can be read-only, write-only, or read-write.
- It can also allow arbitrary random access (seeking forwards or backwards to any location), or only sequential access (for example in the case of a socket or pipe).
- All streams are careful about the type of data you give to them. For example giving a **str** object to the write() method of a binary stream will raise a TypeError

. So will giving a **bytes** object to the write() method of a text stream.

- This module is a part of the standard library, so there's no need to install it separately using pip.

To import the io module, we can do the following:

```
import io
```

In the `io` module there are 2 common classes which are very useful for us:

- **BytesIO** -> I/O operations on byte data
- **StringIO** -> I/O operations on string data

We can access these classes using `io.BytesIO` and `io.StringIO`.

The io.open() method is preferable over os.open() method for file related operations.

Python BytesIO Class

Here, we can keep our data in the form of bytes (`b''`). When we use `io.BytesIO`, the data is held in an in-memory buffer.

We can get an instance to the byte stream using the constructor:

```
import io
bytes_stream = io.BytesIO(b'Hello from Journaldev\x0AHow are you?')
```

To actually print the data inside the buffer, we need to use `bytes_stream.getvalue()`.

```
import io
bytes_stream = io.BytesIO(b'Hello from Journaldev\x0AHow are you?')
print(bytes_stream.getvalue())
```

Here, `getvalue()` takes the value of the byte string from the handle.

Python StringIO Class

Similar to `io.BytesIO`, the `io.StringIO` class can read string related data from a StringIO buffer.

```
import io

string_stream = io.StringIO("Hello from Journaldev\nHow are you?")
```

We can read from the string buffer using `string_stream.read()` and write using `string_stream.write()`. This is very similar to reading / writing from a file!

We can print the contents using `getvalue()`.

```
import io

string_stream = io.StringIO("Hello from Journaldev\nHow are you?")

# Print old content of buffer
print(f'Initially, buffer: {string_stream.getvalue()}')

# Write to the StringIO buffer
string_stream.write('This will overwrite the old content of the buffer if t

print(f'Finally, buffer: {string_stream.getvalue()}')

# Close the buffer
string_stream.close()
```

Output

```
Initially, buffer: Hello from Journaldev
How are you?
Finally, buffer: This will overwrite the old content of the buffer if the l
```

Since we are writing to the same buffer, the new contents will obviously overwrite the old one!

Text I/O

- Text I/O expects and produces str objects. This means that whenever the backing store is natively made of bytes (such as in the case of a file), encoding and decoding of data is made transparently as well as optional translation of platform-specific newline characters.
- The easiest way to to create a text stream is with open(), optionally specifying an encoding:

```
f = open("myfile.txt", "r", encoding="utf-8")
```

In-memory text streams are also available as `StringIO` objects:

```
f = io.StringIO("some initial text data")
```

Binary I/O

- Binary I/O (also called *buffered I/O*) expects and produces bytes objects. No encoding, decoding, or newline translation is performed. This category of streams can be used for all kinds of non-text data, and also when manual control over the handling of text data is desired.
- The easiest way to create a binary stream is with open() with ‘b’ in the mode string:

```
f = open("myfile.jpg", "rb")
```

In-memory binary streams are also available as `BytesIO` objects:

```
f = io.BytesIO(b"some initial binary data: \x00\x01")
```

Raw I/O

- Raw I/O (also called *unbuffered I/O*) is generally used as a low-level building-block for binary and text streams; it is rarely useful to directly manipulate a raw stream from user code. Nevertheless, you can create a raw stream by opening a file in binary mode with buffering disabled:

```
f = open("myfile.jpg", "rb", buffering=0)
```

Encoding

- Unlike other languages, the character 'a' does not imply the number 97 until it is encoded using ASCII (or other equivalent encodings).
- Moreover, the default encoding is platform dependent. In windows, it is 'cp1252' but 'utf-8' in Linux.
- So, we must not also rely on the default encoding or else our code will behave differently in different platforms.
- Hence, when working with files in text mode, it is highly recommended to specify the encoding type.
- f = open("test.txt",mode = 'r',encoding = 'utf-8')

Processing CSV Data : using pandas

- Input as CSV File
  - csv file is a text file in which the values in the columns are separated by a comma.

```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Tusar,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
5,Gary,843.25,2015-03-27,Finance
6,Rasmi,578,2013-05-21,IT
7,Pranab,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

  - Reading a CSV File
    - **read_csv** function of the pandas library is used read the content of a CSV file

```
import pandas as pd
data = pd.read_csv('path/input.csv')
print (data)
```

Processing CSV Data : using pandas

- Reading Specific Rows

```
import pandas as pd
data = pd.read_csv('path/input.csv')

# Slice the result for first 5 rows
print (data[0:5]['salary'])
```

```
0    623.30
1    515.20
2    611.00
3    729.00
4    843.25
Name: salary, dtype: float64
```

- Reading Specific Columns: use the multi-axes indexing method called **.loc()**

```
import pandas as pd
data = pd.read_csv('path/input.csv')

# Use the multi-axes indexing funtion
print (data.loc[:,['salary','name']])
```

	salary	name
0	623.30	Rick
1	515.20	Dan
2	611.00	Tusar
3	729.00	Ryan
4	843.25	Gary
5	578.00	Rasmi
6	632.80	Pranab
7	722.50	Guru

Processing CSV Data : using pandas

- Reading Specific Columns and Rows

```
import pandas as pd
data = pd.read_csv('path/input.csv')

# Use the multi-axes indexing funtion
print (data.loc[[1,3,5],['salary','name']])
```

```
   salary   name
1   515.2    Dan
3   729.0   Ryan
5   578.0  Rasmi
```

```
import pandas as pd
data = pd.read_csv('path/input.csv')

# Use the multi-axes indexing funtion
print (data.loc[2:6,['salary','name']])
```

Working with the CSV Module

- The CSV module includes all the necessary functions built in. They are:
  - csv.reader
  - csv.writer
  - csv.register_dialect
  - csv.unregister_dialect
  - csv.get_dialect
  - csv.list_dialects
  - csv.field_size_limit
- CSV Sample File

```
Title,Release Date,Director
And Now For Something Completely Different,1971,Ian MacNaughton
Monty Python And The Holy Grail,1975,Terry Gilliam and Terry Jones
Monty Python's Life Of Brian,1979,Terry Jones
Monty Python Live At The Hollywood Bowl,1982,Terry Hughes
Monty Python's The Meaning Of Life,1983,Terry Jones
```

Working with the CSV Module

- Reading CSV Files

```
import CSV
With open('some.csv', 'rb') as f:
reader = csv.reader(f)
for row in reader:
print row
```

- Extracting information from a CSV file

```
import csv

f = open('attendees1.csv')
csv_f = csv.reader(f)

for row in csv_f:
  print row[2]
```

Working with the CSV Module

- Writing to CSV Files

```
import csv

csvData = [['Person', 'Age'], ['Peter', '22'], ['Jasmine', '21'], ['Sam', '24']]

with open('person.csv', 'w') as csvFile:
    writer = csv.writer(csvFile)
    writer.writerows(csvData)

csvFile.close()
```

```
Person,Age
Peter,22
Jasmine,21
Sam,24
```

- **Modifying existing rows**

```
import csv

row = ['2', ' Marie', ' California']

with open('people.csv', 'r') as readFile:
reader = csv.reader(readFile)
lines = list(reader)
lines[2] = row

with open('people.csv', 'w') as writeFile:
     writer = csv.writer(writeFile)
     writer.writerows(lines)

readFile.close()
writeFile.close()
```

```
SN, Name, City
1, John, Washington
2, Marie, California
3, Brad, Texas
```

OS Module in Python

- The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.
- **Handling the Current Working Directory**
- Consider **Current Working Directory(CWD)** as a folder, where the Python is operating. Whenever the files are called only by their name, Python assumes that it starts in the CWD which means that name-only reference will be successful only if the file is in the Python's CWD.
  Note: The folder where the Python script is running is known as the Current Directory. This is not the path where the Python script is located.
- To get the location of the current working directory os.getcwd() is used.

```
import os

cwd = os.getcwd()

print("Current working directory:", cwd)
```

Changing the Current working directory

- To change the current working directory(CWD) os.chdir() method is used. This method changes the CWD to a specified path. It only takes a single argument as a new directory path.

```
import os

# Function to Get the current
# working directory
def current_path():
    print("Current working directory before")
    print(os.getcwd())
    print()


# Driver's code
# Printing CWD before
current_path()

# Changing the CWD
os.chdir('../')

# Printing CWD after
current_path()
```

Creating a Directory

- There are different methods available in the OS module for creating a directory. These are –
- os.mkdir()
- os.makedirs()

Using os.mkdir()

- os.mkdir() method in Python is used to create a directory named path with the specified numeric mode. This method raises FileExistsError if the directory to be created already exists.

Using os.makedirs()

- os.makedirs() method in Python is used to create a directory recursively. That means while making leaf directory if any intermediate-level directory is missing, os.makedirs() method will create them all.

Example 1: Create a Directory using Python in a specified location

```
# import library
import os

# this is the directory that will be created
path = "C:/Users/user/Desktop/TestDirectory"

# use os.mkdir() to create the directory
os.mkdir(path)

print("Succesfully created directory " + path)
```

Output

```
Succesfully created directory C:/Users/user/Desktop/TestDi
```

Example 2: Providing Permissions for Reading and Writing Operations Within the Directory

```
# import library
import os

# this is the directory that will be created
path = "C:/Users/user/Desktop/ReadWriteDirectory"

# use os.mkdir() to create the directory
os.mkdir(path, mode = 0o666)

print("Succesfully created directory" + path)
```

Output

```
Succesfully created directory C:/Users/user/Desktop/ReadWr
```

Here, setting mode = 0o666 allows the user to perform read and write file operations within the created directory.

- File permission 755 means **that the directory has the default permissions -rwxr-xr-x** (represented in octal notation as 0755).

The following example shows the usage of makedirs() method.

```
#!/usr/bin/python

import os, sys

# Path to be created
path = "/tmp/home/monthly/daily"

os.makedirs( path, 0755 );

print "Path is created"
```

When we run above program, it produces following result –

```
Path is created
```

Listing out Files and Directories with Python

- ***os.listdir()*** method in python is used to get the list of all files and directories in the specified directory. If we don't specify any directory, then list of files and directories in the current working directory will be returned.

Syntax: *os.listdir(path)*

Parameters:

path (optional) : path of the directory

Return Type: *This method returns the list of all files and directories in the specified path. The return type of this method is list.*

```
# importing os module
import os

# Get the list of all files and directories
# in the root directory
path = "/"
dir_list = os.listdir(path)

print("Files and directories in '", path, "' :")

# print the list
print(dir_list)
```

Deleting Directory or Files using Python

OS module proves different methods for removing directories and files in Python. These are –

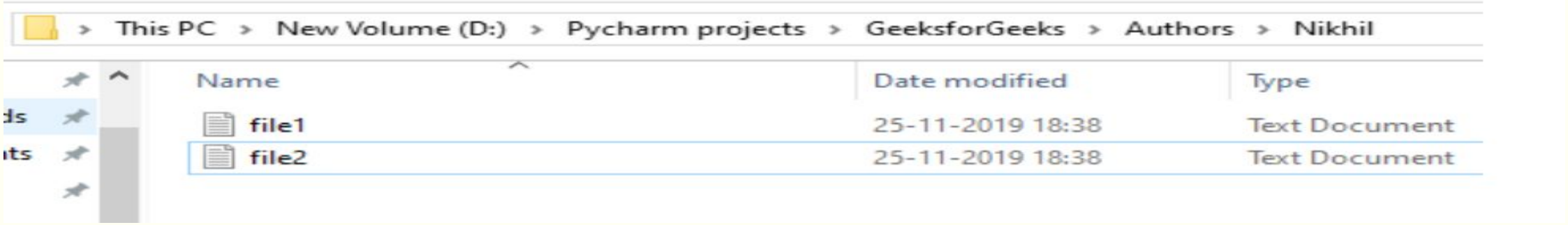
- Using os.remove()
- Using os.rmdir()

Using os.remove()

- os.remove() method in Python is used to remove or delete a file path. This method can not remove or delete a directory. If the specified path is a directory then OSError will be raised by the method.

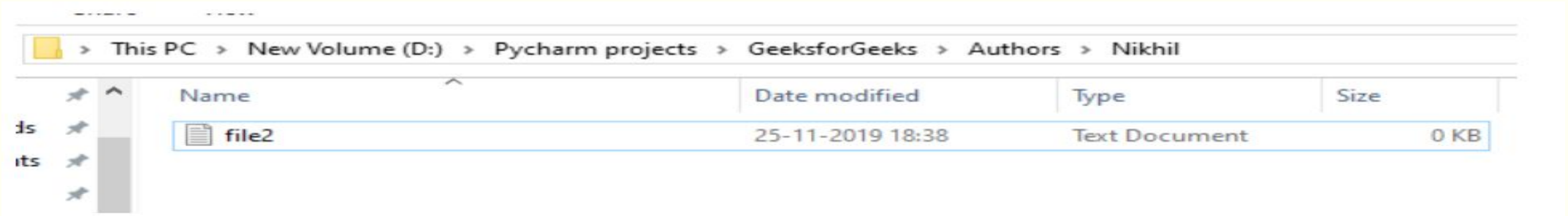
Using os.rmdir()

- os.rmdir() method in Python is used to remove or delete an empty directory. OSError will be raised if the specified path is not an empty directory.



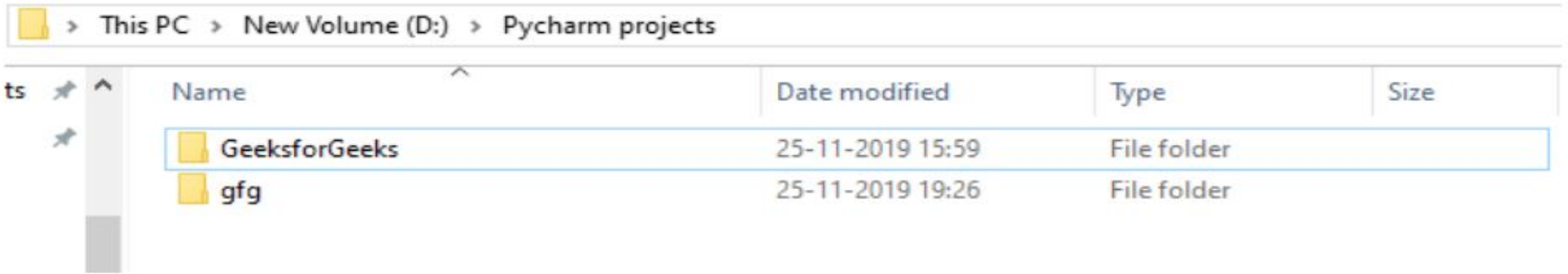
Deleting File

```
import os
file = 'file1.txt'
location = "D:/Pycharm projects/GeeksforGeeks/Authors/Nikhil/"
path = os.path.join(location, file)
os.remove(path)
```




```
# importing os module
import os

# Directory name
directory = "Geeks"

# Parent Directory
parent = "D:/Pycharm projects/"

# Path
path = os.path.join(parent, directory)

# Remove the Directory
# "Geeks"
os.rmdir(path)
```

Deleting Directory



Commonly Used Functions

Sr No	Function	Description
1	**os.name**	This function gives the name of the operating system dependent module imported. The following names have currently been registered: ‘posix’, ‘nt’, ‘os2’, ‘ce’, ‘java’ and ‘riscos’
2	**os.error**	All functions in this module raise OSError in the case of invalid or inaccessible file names and paths, or other arguments that have the correct type, but are not accepted by the operating system. os.error is an alias for built-in OSError exception.
3	**os.popen()**	This method opens a pipe to or from command. The return value can be read or written depending on whether the mode is ‘r’ or ‘w’. os.popen(command[, mode[, bufsize]])
4	**os.close()**	Close file descriptor fd. A file opened using open(), can be closed by close()only. But file opened through os.popen(), can be closed with close() or os.close(). If we try closing a file opened with open(), using os.close(), Python would throw TypeError.
5	**os.rename()**	A file old.txt can be renamed to new.txt, using the function os.rename(). The name of the file changes only if, the file exists and the user has sufficient privilege permission to change the file.
6	**os.remove()**	Using the Os module we can remove a file in our system using the remove() method. To remove a file we need to pass the name of the file as a parameter.
7	**os.path.exists()**	This method will check whether a file exists or not by passing the name of the file as a parameter. OS module has a sub-module named PATH by using which we can perform many more functions.
8	**os.path.getsize():**	In this method, python will give us the size of the file in bytes. To use this method we need to pass the name of the file as a parameter.

OS module

Sr.No	Function Name	Description
1	os.getcwd()	The *getcwd()* method displays the current working directory.
2	os.mkdir("newdir")	use the *mkdir()* method of the **os** module to create directories in the current directory.
3	os.chdir("newdir")	*chdir()* method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.
4	os.remove(file_name)	*remove()* method to delete files by supplying the name of the file to be deleted as the argument.
5	os.rename(current_file_name, new_file_name)	The *rename()* method takes two arguments, the current filename and the new filename.
6	os.listdir()	Display files in current working directory

Practical Programs in Syllabus

4. Exploring Files and directories

- a. Python program to append data to existing file and then display the entire file.
- b. Python program to count number of lines, words and characters in a file.
- c. Python program to display file available in current directory

KEEP CALM AND LEARN PYTHON