File Handling in Python

File Handling In Python - Read Write Open Close Files In Python

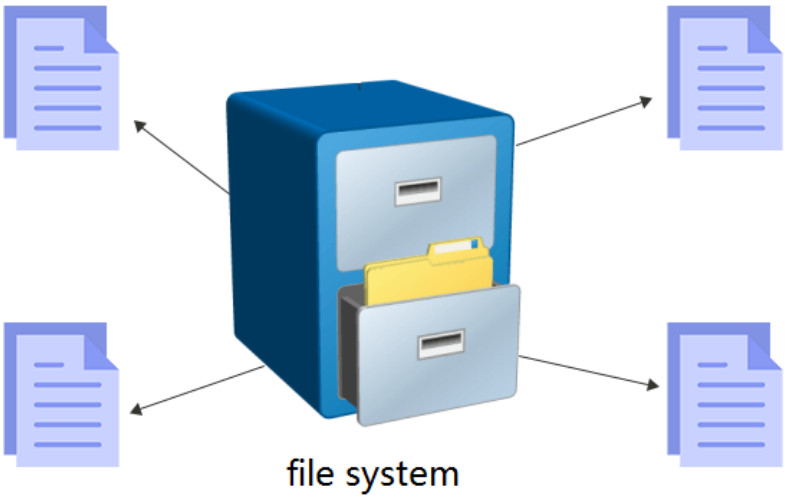
Heres Main Takeaways are :

- **What is File & File Handling**
- **Types Of Files & File Path Supported By Python**
- **Basic File Operations**
- **Importance of with keyword in file handling**
- **Methods of File objects**
- **Python Directory and File Management**

What is File ? :

- **File is named location on disk (hard drive) to store related information.**
- **They are used to permanently store data in a non-volatile memory (e.g. hard disk)**

- **Data in all systems in stored in the form of files**
- **We use files for future use of the data by permanently storing them.**
- **A file is the collection of data stored on a disk in one unit identified by filename.**



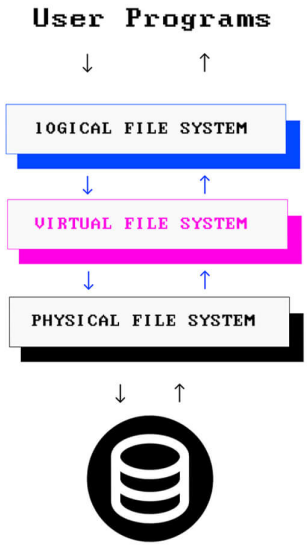
What is Python File Handling ? :

- **File handling is basically the management of the files on a file system.**
- **Every operating system has its own way to store files.**
- **Python File handling is useful to work with files in our programs**
- **We don’t have to worry about the underlying operating system and its file system rules and operations.**

Why Do You Need File Handling ?:

- **Data Persistence i-e keeping data safe for future use is the requirement of every software application.**
- **The data in python program is only available when program is in execution**
- **We can save this data in computer, available even after program exit through File Handling in Python**
- **Imagine You have a server up and running and you need to access files present on that server**

- **You can do this remotely or you need to work with files locally by giving some input to your python program on the server or so.**



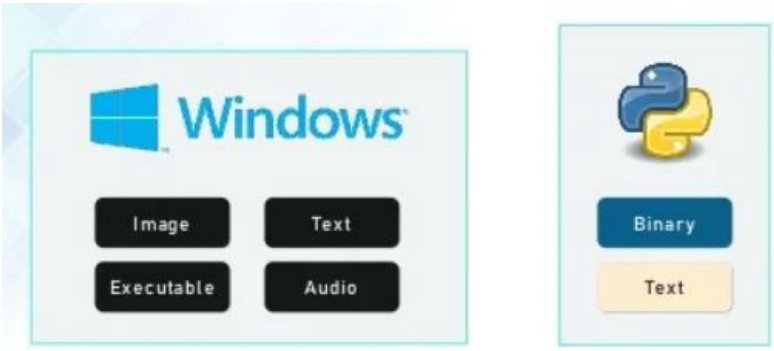
Types Of Files Supported By Python :

There are 2 types of files mainly Supported By Python :

- **Text File : Text files are structured as a sequence of lines, where each line includes a sequence of characters. For example, test.txt**

- **Binary File: A binary file is any type of file that is not a text file which are used to store binary data such as images, video files, audio files, etc.**

Binary files are categorized as the generic 0's and 1's in Python too.

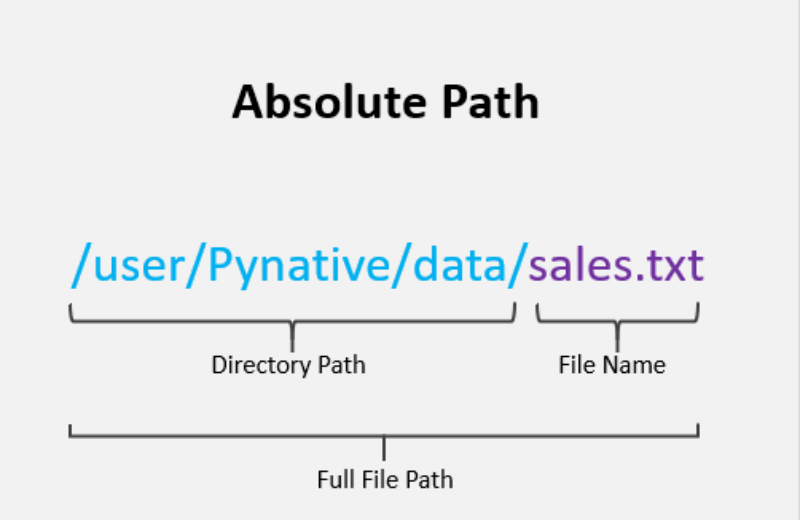


Types Of Files Supported By Python :

A file path defines the location of a file or folder in the computer system. There are two ways to

specify a file path :

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



Basic File Operations

- In file handling when we want to read a file from or we want to write a file first we need to open that file first
- And when we are done it needs to be closed, so that recourses that are tied with the file are freed.

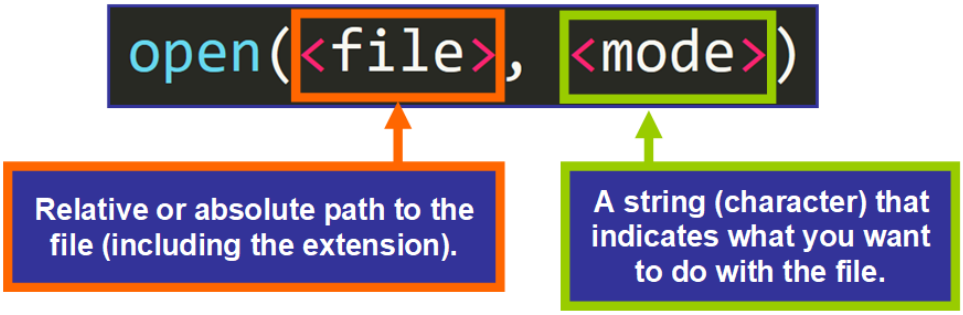
Following are the basic operations in File Handling in Python:-

- Open File in Python
- Read File in Python
- Write in File

- **Append to File**
- **Closing File in Python**

OPEN A FILE: :

- **The key function for working with files in python is open ( ) function**
- **The open method returns a file object which is used to access the write, read and other in-built methods.**



- **The open ( ) function has two parameters. They are :**
- **Filename : It is the name of the file with path**
- **Mode : It tells the program in which mode the file has to be open.**

Modes:

It really makes sense for Python to grant only certain permissions based what you are planning to do with the file

Modes available are :

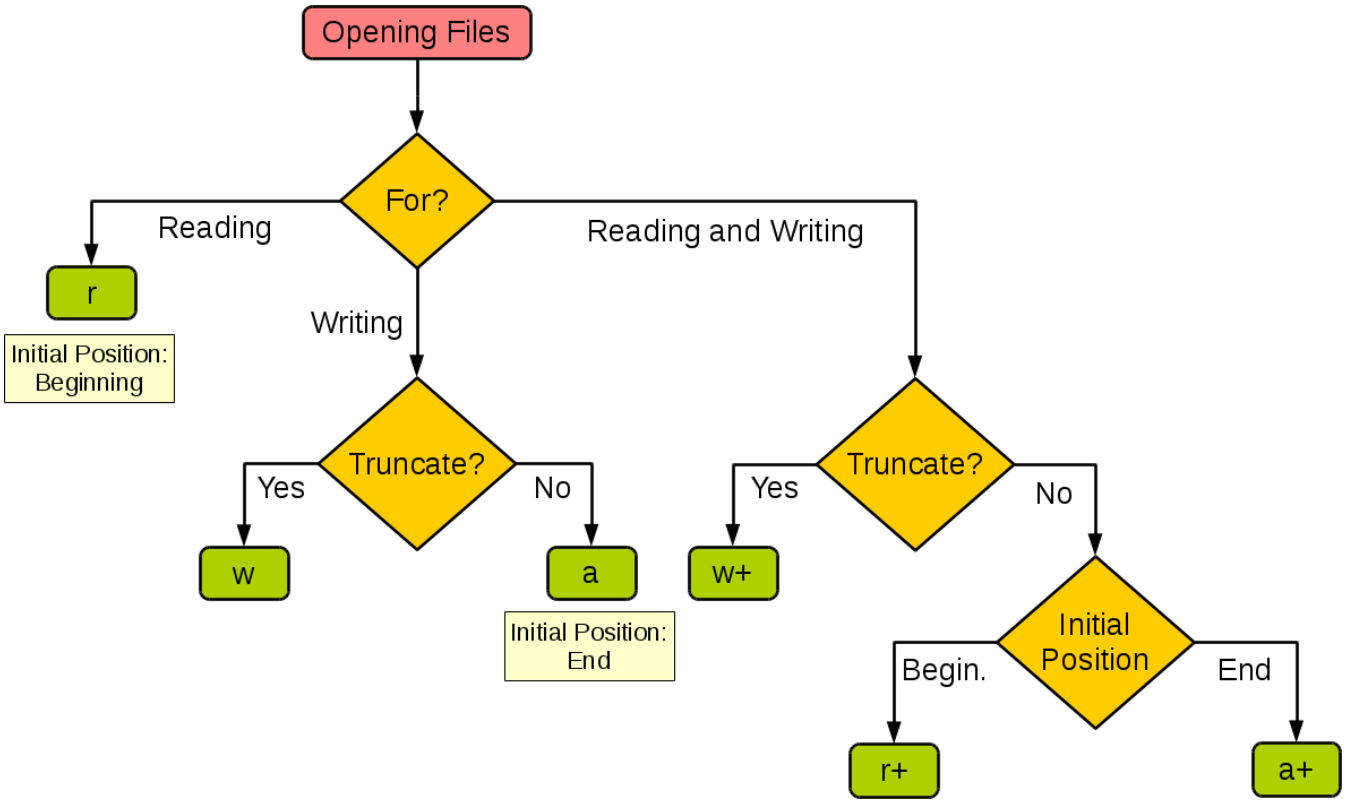
- **Read ("r").**
- **Append ("a")**
- **Write ("w")**
- **Create ("x")**

We can also choose to open the file in:

- **Text mode ("t")**

- **Binary mode ("b")**

To be able to read a file and perform another operation in the same program, you need to add the "+" symbol to the mode



In [1]:
```
#Opening the file
demo_file = open('Demo.txt', 'r')
#close the file, very important
print(type(demo_file))
demo_file.close()
```

```
<class '_io.TextIOWrapper'>
```

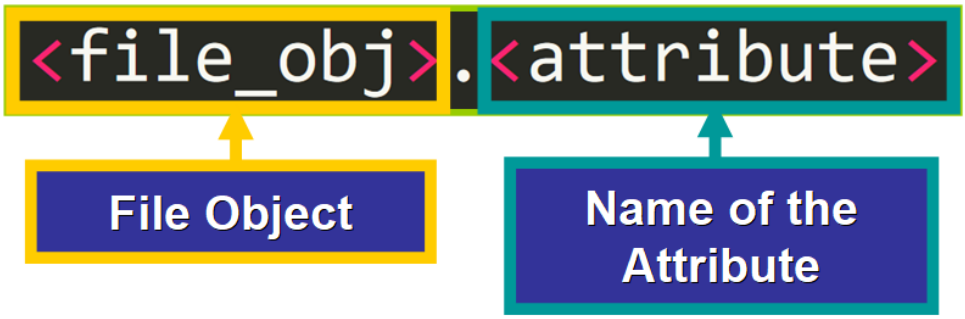
File Objects :

- **File Object is an object exposing a file-oriented API (with methods such as read() or write()) to an underlying resource.**

- **A file object is an object that lets us work and interact with existing files in our Python program.**

- **File objects have attributes, such as:**

```
name: the name of the file.
closed: True if the file is closed. False otherwise.
mode: the mode used to open the file.
```



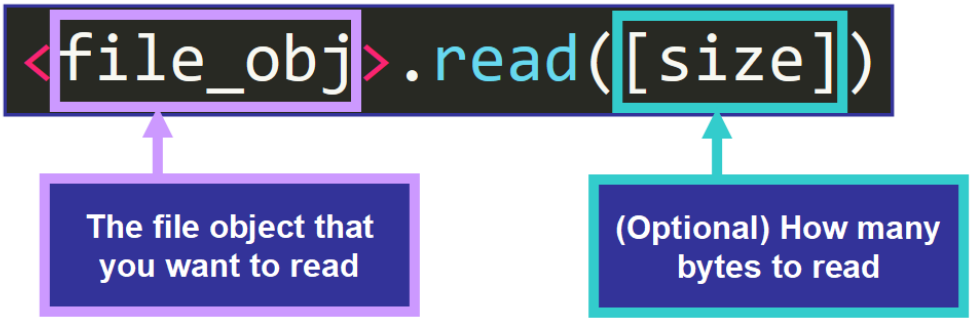
read ( ) function :

The read ( ) function is used to read the contents of the file.

In [2]:
```
demo_file = open("Demo.txt", "r")
print(demo_file.read())
demo_file.close()
```

```
Open is used to open the file
Read is used to read the file
```

we can also read characters of file from read ( ) method.



In [3]:
```
demo_file = open("Demo.txt", "r")
print(demo_file.read(5))
#if we now use the read() method again then it start reading from the current location o
print(demo_file.read(10))
```

```
Open
is used to
```

current cursor location can be changed using seek().

tell() method is use to know the current position of cursor.

In [4]:
```
print(demo_file.tell())
print(demo_file.seek(0))
print(demo_file.tell())
```

```
15
0
0
```

In [5]:
```
print(demo_file.read(15))
```

```
Open is used to
```

we can also read a file line by line using a for loop.

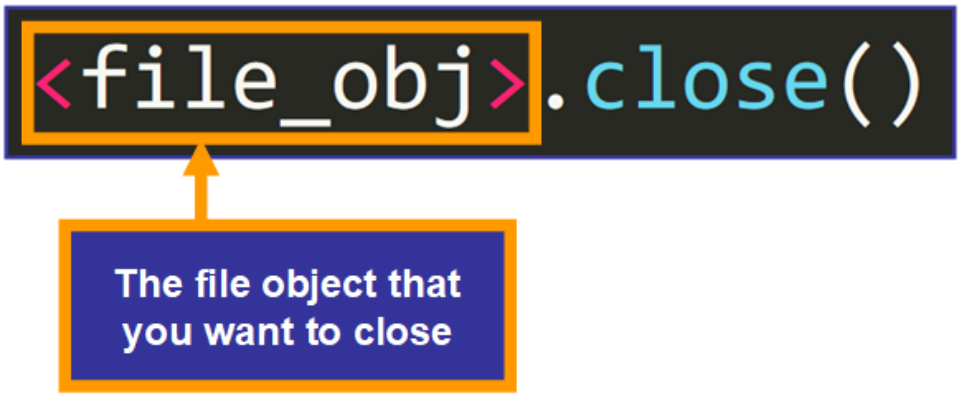
In [6]:
```
demo_file.seek(0)
for line in demo_file:
    print(line)
```

```
Open is used to open the file

Read is used to read the file
```

close ( ) function

- **The close ( ) function is used to close a particular file post manipulations on it.**
- **After writing to a file, if we do not call the close() method, all the data written to the file will not be saved in it.**
- **It's always a good idea to close the file after we are done with it to release the resources.**

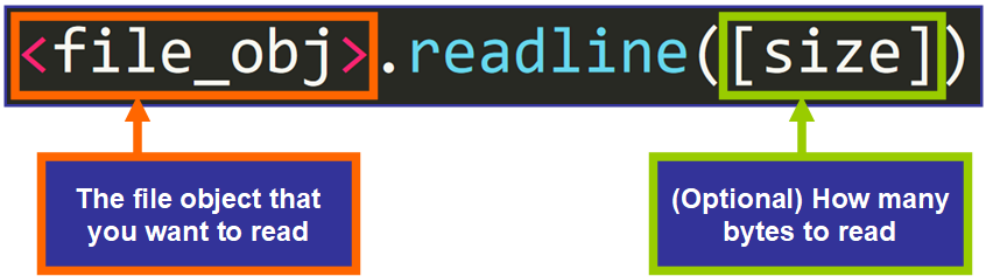


In [ ]:
```
demo_file.close()
```

Readline ( ) vs. Readlines ( )

We can also read a file line by line with Readline ( ) and Readlines ( ) Methods .

- **readline ( ) : readline ( ) reads one line of the file until it reaches the end of that line**
- **readlines ( ) : In contrast, readlines ( ) returns a list with all the lines of the file as individual elements (strings)**



In [10]:
```
demo_file = open("Demo.txt", "r")
print(demo_file.readline())
print(demo_file.readlines())
demo_file.close()
```

```
Open is used to open the file

['Read is used to read the file\n', 'Readlines reading this line']
```

write ( ) function:

Sometimes, you may want to delete the content of a file and replace it entirely with new content. You can do this with the write ( ) method if you open the file with the "w" mode.

In [11]:
```
#write function - mode = w+ -Read and Write
demo_file = open('Demo.txt','w')
demo_file.write("When we use write previous data is earased.\n")
demo_file.write("And new data is appended.")
demo_file.close()
```

In [12]:
```
demo_file = open("Demo.txt", "r")
print(demo_file.read())
demo_file.close()
```

```
When we use write previous data is earased.
And new data is appended.
```

Append :

Appending means adding something to the end of another thing. The "a" mode allows you to open a file to append some content to it.

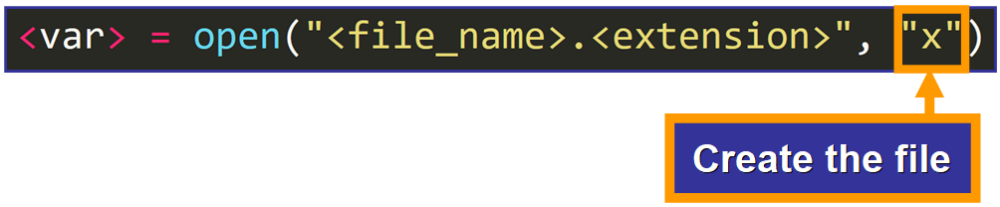
In [13]:
```
demo_file = open('Demo.txt','a')
```

```
demo_file.write("\nStatement added to the end of the file..")
demo_file.close()
```

In [ ]:
```
demo_file = open("Demo.txt", "r")
print(demo_file.read())
demo_file.close()
```

Create a File

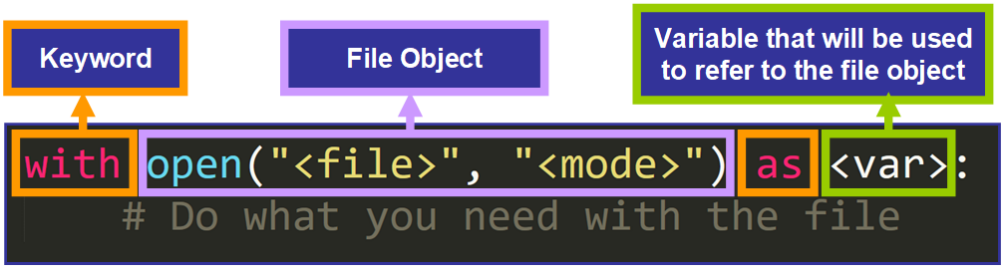
If we need to create a file "dynamically" using Python, we can do it with the "x" mode.



In [14]:
```
#Creating a file
f = open("new_file.txt", "x")
```

Context Managers

- **By using Context Manager -- with Keyword you don't need to remember to close a file at the end of your program**
- **And we have access to the file in the particular part of the program that you choose.**
- **The body of the context manager has to be indented, just like we indent loops, functions, and classes.**



In [15]:
```
with open("demo.txt", "r") as f:
    print(f.readline())
```

```
When we use write previous data is earased.
```

In [16]:
```
with open("demo.txt", "r") as f:
    print(f.readline())
#demo.txt.closed()
```

```
# Trying to read the file again, outside of the context manager
print(f.readlines())
```

```
When we use write previous data is earased.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [16], in <cell line: 4>()
      2     print(f.readline())
      3 # Trying to read the file again, outside of the context manager
----> 4 print(f.readlines())

ValueError: I/O operation on closed file.
```

Handle Exceptions When Working With Files

- **FileNotFoundError : Raised when a file or directory is requested but doesn't exist.**
- **PermissionError : Raised when trying to run an operation without the adequate access rights - for example filesystem permissions.**
- **IsADirectoryError : Raised when a file operation is requested on a directory.**

In [17]:
```
f = open("names.txt")
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [17], in <cell line: 1>()
----> 1 f = open("names.txt")

FileNotFoundError: [Errno 2] No such file or directory: 'names.txt'
```

In [18]:
```
#file not foind error
try:
    f = open("names.txt")
except FileNotFoundError:
    print("The file doesn't exist")
```

```
The file doesn't exist
```

Python Directory and File Management :

In [20]:
```
import os
#shutil , psutil
```

In [21]:
```
#Get current Directory
os.getcwd()
```

Out[21]:
```
'C:\\Users\\nithe\\OneDrive\\Desktop\\TNR\\Data Science\\Course Bundle\\01 - Python\\15 - File Handling'
```

In [22]:
```
#Changing Directory
os.chdir("/Users/nithe/OneDrive/Desktop")
os.getcwd()
```

Out[22]:
```
'C:\\Users\\nithe\\OneDrive\\Desktop'
```

In [23]:
```
#Making New Directory
os.mkdir("hey")
```

In [24]:
```
#removing directory
os.rmdir("hey")
```

In [26]:
```
#Creating a Directory
os.mkdir('sample')

#Changing the directory
os.chdir('./sample')

#creating a file
f=open('testfile.txt','x+')

#writing to the file
f.write("hello world")

#Reading the flie without closing
print(f.read())

#Closing the flie
f.close()
```

In [27]:
```
#reading a flie after closing using context manager
with open("testfile.txt", "r") as f:
    print(f.read())
```

```
hello world
```

In [28]:
```
#Renaming a file
os.rename("testfile.txt","sample.txt")
```

In [29]:
```
f=open('sample.txt','r')
f.readline()
```

Out[29]:
```
'hello world'
```

In [30]:
```
#removing a file
f.close()
os.remove("sample.txt")

f=open("sample.txt",'r')
f.readline()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [30], in <cell line: 4>()
      2 f.close()
      3 os.remove("sample.txt")
----> 4 f=open("sample.txt",'r')
      5 f.readline()

FileNotFoundError: [Errno 2] No such file or directory: 'sample.txt'
```

FAQ's on Flie Handling:

- **Can you explain the different modes of opening a file in Python ?**
- **How do you create a text file using Python ?**
- **How do you read and write to an existing file in Python ?**
- **Is it possible to open multiple files using Python ? If yes, then how ?**
- **How should you handle exceptions when dealing with files in Python ?**
- **What are some important methods used for reading from a file in Python ?**
- **What are some common errors that can occur while working with files in Python ?**
- **What's the difference between binary and text files in Python ?**