**File Handling:**

* In computer terms, “a file is a resource for saving data and information in computer hardware”.
* A file is opened in the RAM, but it is stored in the hardware because the hardware because hardware is non-volatile i.e., it stores its data permanently.
* RAM is volatile, it loses its data when the system is shutdown.
* Unlike C or C++, file handling in Python is relatively easy and simple.
* Python treats files differently **as text or binary** and this is important.

**There are two types of file:**

1. **Text file**
2. **Binary file**

* Name of the text file that it must contain text in it. The extension for the text file is .txt .
* All other forms of the files are mostly binary even a .doc file, that we open in Microsoft Word is a binary file because it requires special software for accessing it.
* Second sort of files are binary files. They are almost all other files that we come in contact with while using with while our computer.
* These files include images, PDFs, Excel files, etc.

**Modes of opening file in Python:**

There are many modes of opening a file in Python, unlike other languages Python has provided its users a variety of options.

* r: r mode opens a file for read-only. We do not have permission to update or change any data in this mode.
* w: w mode does not concern itself with what is present in the file. It just opens a file for writing and if there is already some data present in the file, it overwrites it.
* x: x is used to create a new file. It does not work for an already existing file, as in such cases the operation fails.
* a:  a stands for append, which means to add something to the end of the file. It does exactly the same. It just adds the data we like in write(w) mode but instead of overwriting it just adds it to the end of the file. It also does not have the permission of reading the file.
* t: t mode is used to open our file in text mode and only proper text files can be opened by it. It deals with the file data as a string.
* b: b stands for binary and this mode can only open the binary files, that are read in bytes. The binary files include images, documents, or all other files that require specific software to be read.
* +: In plus mode, we can read and write a file simultaneously. The mode is mostly used in cases where we want to update our file.

**open(), read(), readline() & readlines() for Reading File:**

* When we open a file, we are asking the operating system to find the file by name, making sure the file exists.

**How to open a file?**

* Python has a built-in open() function to open a file.
* The syntax of the function is:
* open(“filename” ,”mode”)

**To open a file, we must specify two things:**

1. Name of the file and its extension
2. Access mode where we can specify in which mode file has to be opened, it could either be read (r), write (w) or append(a), etc. For more information regarding access modes.

* Example: open(“myfile.txt”)
* The file “myfile.txt” will open in “rt” mode as it is the default mode. But the best practice is to follow the syntax to avoid errors.

**Use file object to maintain file:**

* On opening function returns a file object. We store this file object into a variable which is generally called file pointer/file handler.
* Example: **f=open(“myfile.txt”,”w”)**
* we can use this file pointer to further add modifications in the file.
* An error could also be raised if the operation fails while opening the file.
* It could be due to various reasons like trying to access a file that is already closed or trying to read a file open in write mode.

**How to read a file?**

For reading a file in python, there are various methods available

* We can read a whole file line by line using a for loop in combination with an iterator.
* When opening a file for reading, python needs to know exactly how the file should be opened. Two access modes are available reading(r ), and reading in binary mode (rb). They have to be specified during opening a file with the built-in open() method.
* Example : f=open(“myfile.txt”,”r”)
* read() method reads the whole file by default. We can also use the read(size) method where you can specify how many characters we want to return i.e.
* Example: ***f.read(2) # here, you will get the first two characters of the file***.
* We can use the ***readline()*** method to read individual lines of a file. By calling readline() a second time, you will get the next line.
* ***f.readline() # return a single line***
* readlines() method reads until the end the file ends and return a list of lines of the entire file. It does not read more than one line.
* f.readlines() #returns a list object

**Why it is necessary to close the file?**

It is always the best practice to close a file after you are done performing operations on it. However, Python runs a garbage collector to clean up the unused objects, but as good programmers, we must not rely on it to close the file. Python has a build-in close() function to close a file i.e;

* **f.close()**

**Readline() function:**

# File operation with reading each line and writing each line .

'''

 #First file creation and writting .

fo = open ( "first31.txt ", "w")

#fo=open("first.txt","r+")

seq= [ "First Line \n ", "Second Line \n" , "Third Line \n" ,"Fourth Line \n " ]

#,"Fifth line \n "\n,"sixth line "\n , "seventh line \n"]

fo.writelines(seq)

fo.close()

'''

# Open the file in read mode .

fo = open ("first31.txt" , "r")

#lines=fo.readlines()

#print("readlines():",lines)

line1=fo.readline()

print("readline():",line1)

#Below line of code will go to next line and will read how many characters to read.

line2=fo.readline(5)

print("readlines(1):",line2)

# close of the file .

fo.close()

**readlines() function:**

f = open("harry.txt", "rt")

print(f.readlines())

# print(f.readline())

# print(f.readline())

# print(f.readline()) #read single line

# content = f.read() #read whole content

#f.readlines() # read whole line put into list

# for line in f:

# print(line, end="")

# print(content)

# content = f.read(34455)

# print("1", content)

#

# content = f.read(34455)

# print("2", content)

f.close()

'''f1=open("shiva\_file1.txt")  
count=0  
for new\_line in f1:  
 count=count+1  
print(count)  
f1.close()'''  
  
'''f1=open("shiva\_file1.txt",'a')  
f1.writelines("What do you want\n what do you want to try")  
f1.close()'''  
  
'''f1=open("shiva\_file1.txt",'r')  
lines=f1.read()  
lines.lower()  
num=lines.count('a')+lines.count('b')+lines.count('c')  
print(num)  
f1.close()'''

**Writing And Appending to a File:**

* **“w”:** Here “w” stands for write. After opening or creating a file, a function, f.write() is used to insert text into the file. The text is written inside closed parenthesis surrounded by double quotations. There is a certain limitation to the write mode of the opening file that it overrides the existing data into the file. For a newly created file, it does no harm, but in case of already existing files, the previous data is lost as f.write() overrides it.
* **“a”:** “a” symbolizes append mode here. In English, appends mean adding something at the end of an already written document, and the same is the function the mode performs here. Unlike write mode, when we use "a" keyword, it adds more content at the end of the existing content. The same function i.e., f.write() is used to add text to the file in append mode. It is worth noting that append mode will also create a new file if the file with the same name does not exist and can also be used to write in an empty file.
* **“r+”:** At the beginning of the description, I told you that we would learn reading and writing a file simultaneously. Well, r+ mode is more of a combination of reading and append than read and write. By opening a file in this mode, we can print the existing content on to the screen by printing f.read() function and adding or appending text to it using f.write() function.

**Note: 1)** If you are writing in append mode, start your text by putting a **blank space** or **newline character (\n)** else the compiler will start the line from the last word or full stop without any blank space because the curser in case of append mode is placed right after the last character. So, it is always considered a good practice to adopt certain habits that could help you in the future, even though they are not much helpful now.

2) f.close() is used to close a file when we are done with it. It is a good practice to close a file after use because whichever mode you opened it for, the file will be locked in for that specific purpose and could not be accessed outside the program, even though the file browser.

**Write() vs writelines():**

* Write() function is used to add string in a file.
* But with writelines we can pass multiple string in file by passing list .
* file1 = open("Employees.txt", "w")  
  lst = "shiva"  
  '''for i in range(3):  
   name = input("Enter the name of the employee: ")  
   lst.append(name + '\n')'''  
    
  file1.writelines(lst)  
  file1.close()  
  print("Data is written into the file.")
* file = open("Employees.txt", "w")

for i in range(3):

   name = input("Enter the name of the employee: ")

   file.write(name)

   file.write("\n")

file.close()

print("Data is written into the file.")

**Code:**

# f = open("harry.txt", "w")

# a = f.write("Harry bhai bahut achhe hain\n")

# print(a)

# f.close()

# f = open("harry2.txt", "a")

# a = f.write("Harry bhai bahut achhe hain\n")

# print(a)

# f.close()

# Handle read and write both

f = open("harry2.txt", "r+")

print(f.read())

f.write("thank you")

**seek(), tell() function in file handling:**

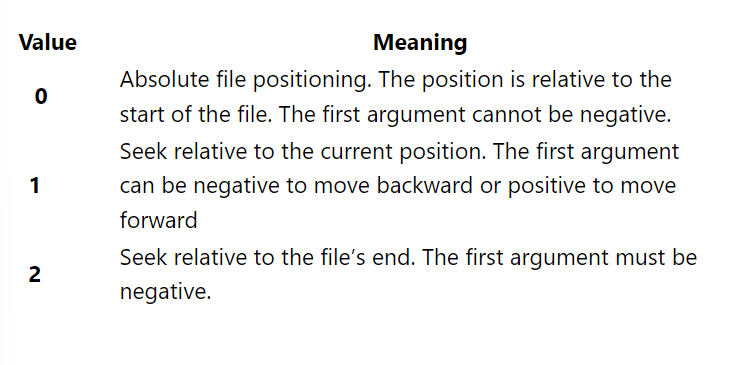
* Python file objects give us many methods and attribute that we can use to analyze a file, including tools to figure out the name of the file associated with the file object, whether it is closed or opened, writable, readable and how it handles errors.

What if we want to know the position of the file(read/write) pointer:

* we use **the tell() function**. f.tell() returns an integer giving the file pointer current position in the file represented as a number of bytes.
* File Pointer/File Handler is like a cursor, which defines from where the data has to be read or written in the file.
* f = open("myfile.txt", "r")
* print(f.readline() )
* print(f.tell())

seek() function:

* syntax: seek()
* parameter required: No parameter are required..
* Return Value: seek() function returns the current position of the file pointer within the file.
* When we open a file, the system points to the beginning of the file. Any read or write will happen from the start. To change the file object’s position, use seek(offset, whence) function. The position will compute by adding offset to a reference point, and the whence argument selects the reference point. It is useful when operating over an open file. If we want to read the file but skip the first 5 bytes, open the file, use function seek(5) to move to where you want to start reading, and then continue reading the file.
* **Syntax:**  file\_pointer .seek(offset, whence).
* **Offset:**   In seek() function, offset is required. Offset is the position of the read/write pointer within the file.
* **Whence:** This is optional. It defines the point of reference. The default is 0, which means absolute file positioning.



f = open("harry.txt")

f.seek(11)

print(f.tell())

print(f.readline())

# print(f.tell())

print(f.readline())

# print(f.tell())

f.close()

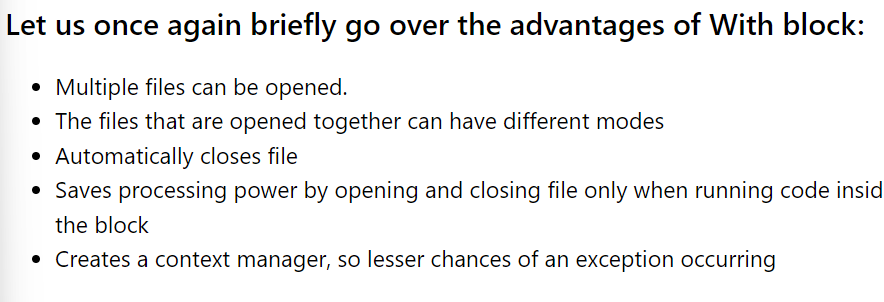
**Using with Block to Open python files:**

* We will see how we can use, with block to open and close a file, including syntax and benefits. **We will be using f as our file’s object**.
* Opening and closing of files are necessary and crucial steps in file handling. We cannot read, write, or perform any task on a file without opening it first.
* If we do not close our file after we are done using it, then the file object will keep on consuming processor memory, and also, there will be more chances of exceptions as the file is still open hence, more chances of bugs.
* To save ourselves from such situations, we could use a with.

**How the with block works?**

With open(“file\_name.txt”) as f:

* **f** being the object of the file
* The important thing to note is, there is no **close() function** required.
* “With block” is that we can open multiple files in a single block by separating them using a comma.
* With open(“file1txt”) as f, open(“file2.txt”) as g

****

with open("harry.txt") as f:

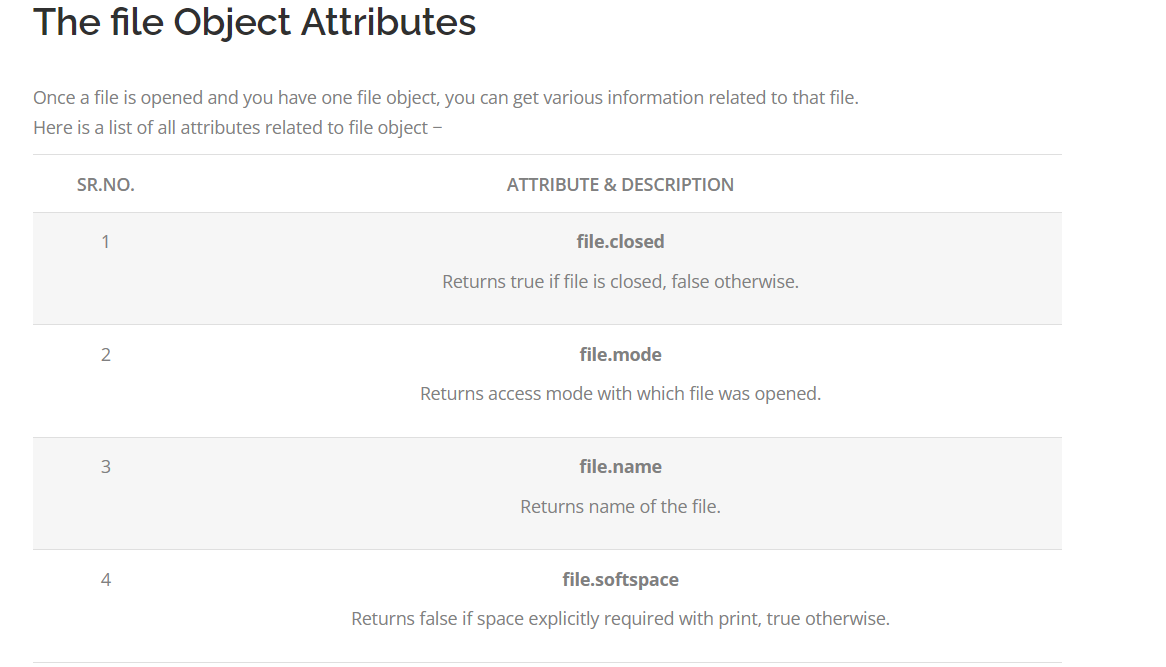
a = f.readlines()

print(a)

# f = open("harry.txt", "rt")

#Question of the day - Yes or No and why? ->No it cannot be open

# f.close()

****

**File handling attributes:**

# File handling attributes

fo = open ("abc.txt", "w")

print ( "Name of the file :",fo.name)

#fo.close()

print ( "Closed or not:", fo.closed)

print ( "Opening mode : ",fo.mode )

import os

#os.rename("newabc.txt","new2abc.txt")

os.remove("abc.txt")

print("Before chdir",os.getcwd())

#os.mkdir("Newdirectory")

os.chdir("Newdirectory")

print("After changing folder:",os.getcwd())

os.mkdir("newfolder")

print(os.rmdir("newfolder"))

**File handling binary file:**# File handling Byte File.

fileobj=open("image.jpg","rb")

filecontent=fileobj.read()

print(filecontent)

fileobj.close()

'''

with open("image.jpg",mode="rb") as file:

    filecontent=file.read()

    print(filecontent)

with open("creatingwrite.txt",mode='w+')as file2:

    filec=file2.write("This is first line.")

    print(file2.read())

'''