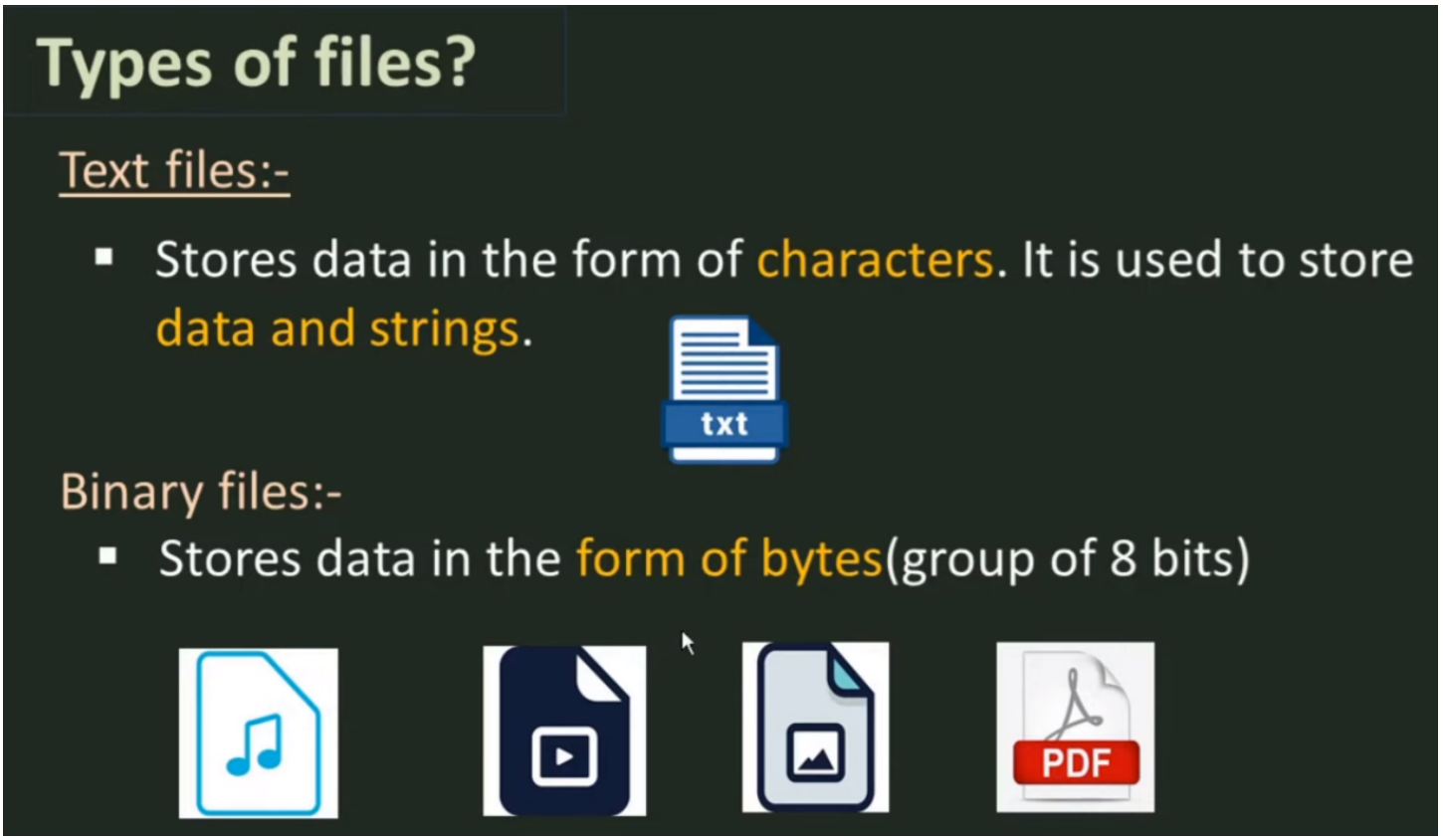
## File Handling

A file is a named area in memory that's set aside to store information. This data is kept in non-volatile memory, meaning it stays there even when the computer is turned off. Files allow us to save and find data whenever we need it.




In file handling, we open a file, perform an action, and then close it. We can store data permanently in two ways:

1. Using File handling 2. Database

Therefore, we need file handling to store data permanently.

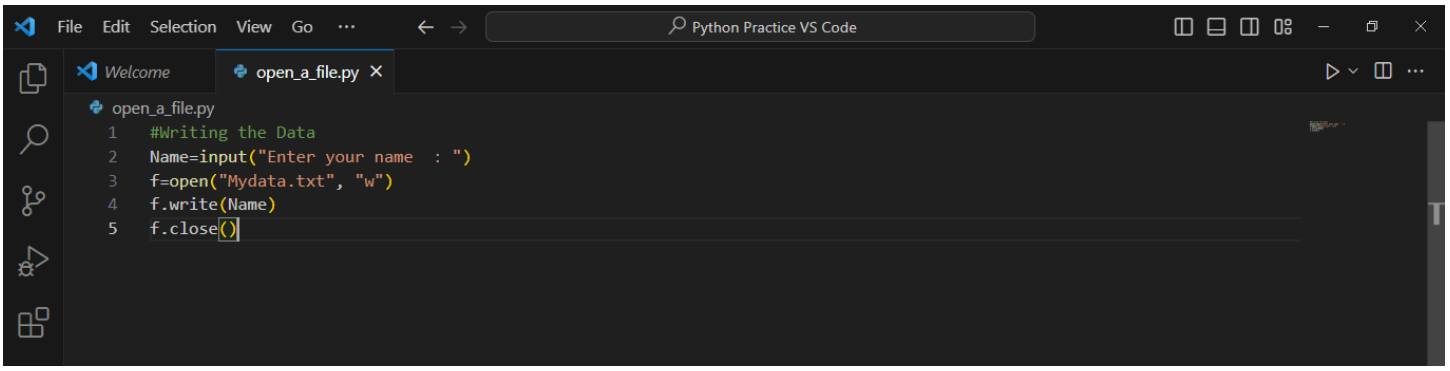
## Opening A File

To open a file we use the built-in open() function, the syntax of this function is as follows:

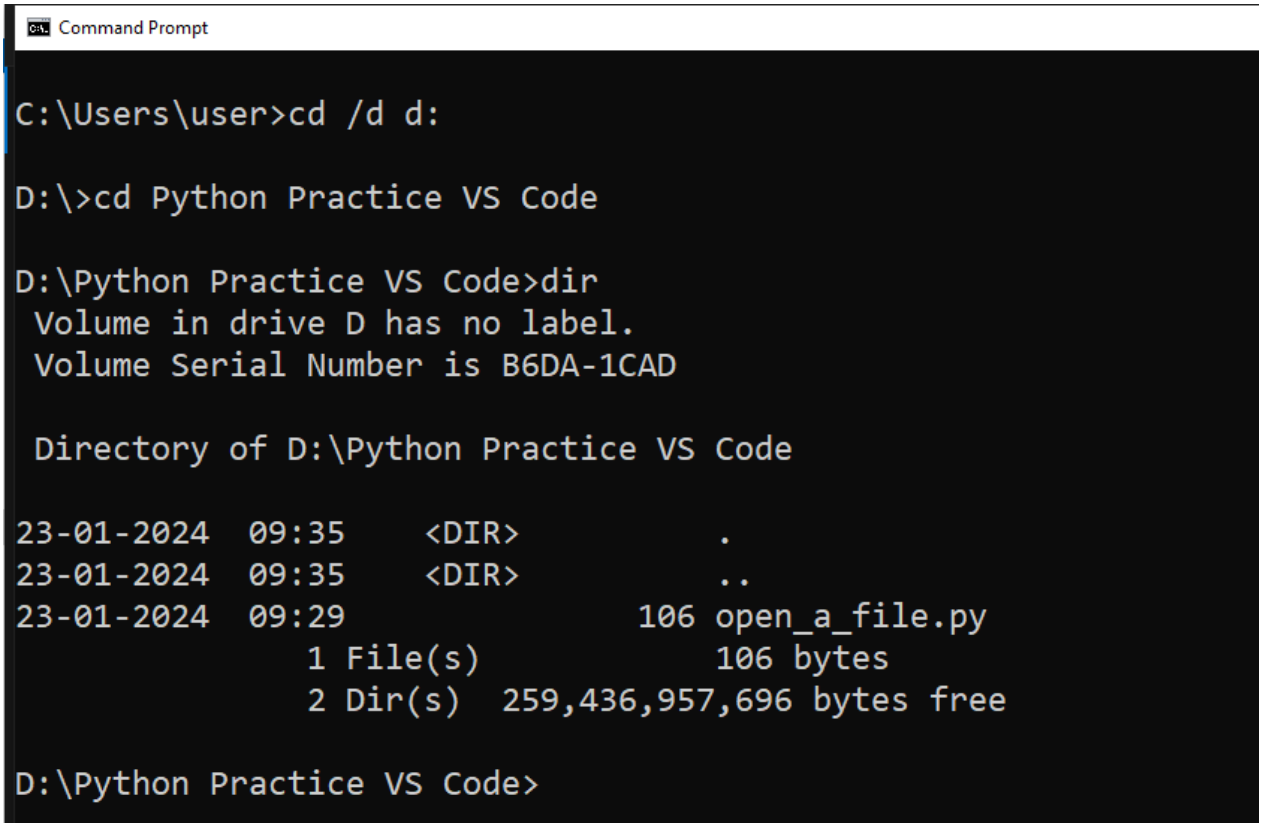
f=open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

Here,
f refers to file pointer, it points to the beginning of a file, file refers to the file path or file name, and mode refers to access mode(purpose of opening a file) and the default access mode is 'r', reading.
**Example**

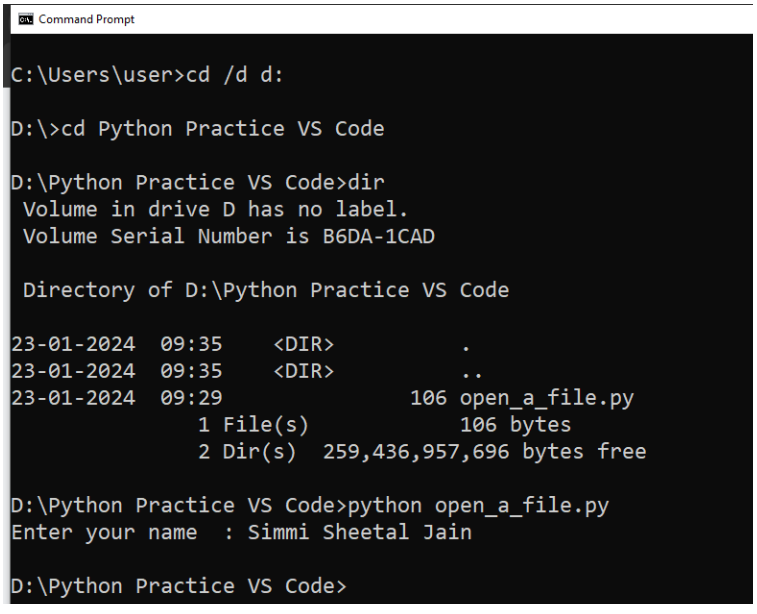
Step 1: Create a separate folder for file handling, then create a file handling program in vs code. Now, save this as a py file.



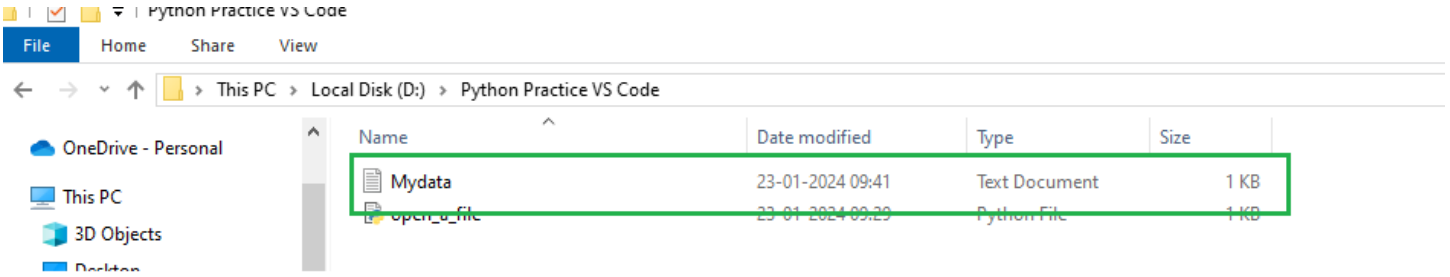

Step 2: Please open the command prompt and proceed to change the current directory. Navigate to the folder that has been created specifically for file handling.



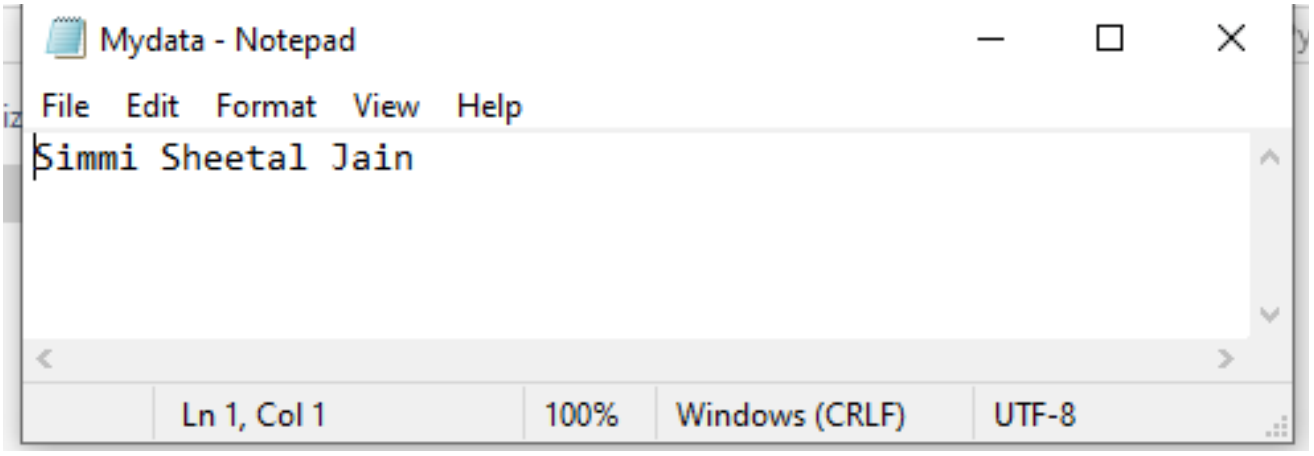

Step 3: Run the file using the python command. See below figure:



Now, check the file has been created in your respective folder.

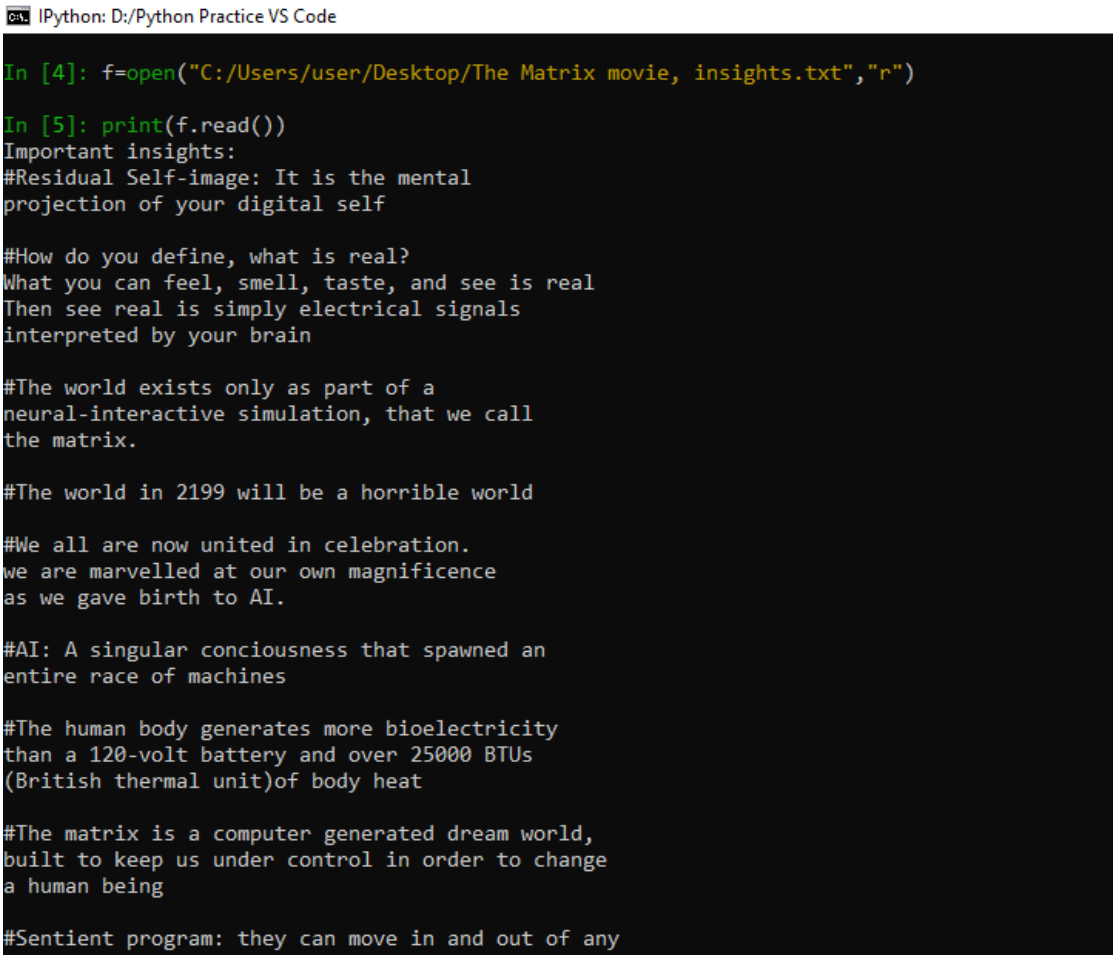



You can open and check the file has data that you have entered:




## Opening a file stored at a different location

To open a file stored at a different location, we need to specify the complete path of the file we want to open. But we have to remove the backward slash present in the file path with the forward slash as shown below:



## Buffering Argument

- Buffering is the third argument in the open()  function. We use it to set the buffer size for the file.
- In text mode, we keep the buffer size more than or equal to 1. But in the binary mode, the buffer size can be zero
- It is an optional argument. The default size of the buffer is 4096-8192 bytes.

## Encoding Argument

- We set an encoding type which is used to decode and encode the file.
- We use it only in text mode
- The Default value of the encoding argument depends on OS. For Windows, it is cp1252

## Error Argument

- This argument in open() represents how encoding and decoding errors will be handled.
- We cannot use it in binary mode
- Some standard values of error argument are: strict, ignore, replace, etc.

## Newline

- New line character can be \n, \r, \r\n, etc.
- New line character depends on the type of file we are opening.

# Closing a file

We use the close() function to close a file.

**Syntax:**

```
file_pointer.close()
```

The close function is responsible for closing the file and deleting the file pointer. If we neglect to close the file manually, the Python garbage collector will eventually handle the task during the program's execution. However, relying on the Python garbage collector to close the file introduces the risk of file data corruption and may result in unnecessary memory wastage. It's recommended to explicitly close the file using the close function to ensure proper file handling and prevent potential issues during the execution of the program.

## File Class and File Object

File class is a built-in class in Python. This class is used to create a file object and then we use this file object to access different variables and methods of this file class.

Here are some common instance variables of the file class:

**name:** Represents the name of the file.

**mode:** Indicates the mode in which the file was opened ('r' for read, 'w' for write, 'a' for append, etc.).

**encoding:** Specifies the encoding of the file.

**closed:** A boolean attribute indicating whether the file is closed or not.

**errors:** Specifies the error handling scheme for encoding and decoding errors.

Here's an example of how you might access these attributes:

Syntax:

file_object.variable_name

Coding Example :

```
In [6]: f.name
Out[6]: 'C:/Users/user/Desktop/The Matrix movie, insights.txt'

In [7]: f.closed
Out[7]: False

In [8]: f.encoding
Out[8]: 'cp1252'

In [9]: f.mode
Out[9]: 'r'

In [10]:
```

## File Object Methods:

The open() function returns a file object. We have the following file object methods in file class:

1. readable()
2. writable()

1. readable(). This method returns ‘True’ if the file is readable else it returns ‘False’

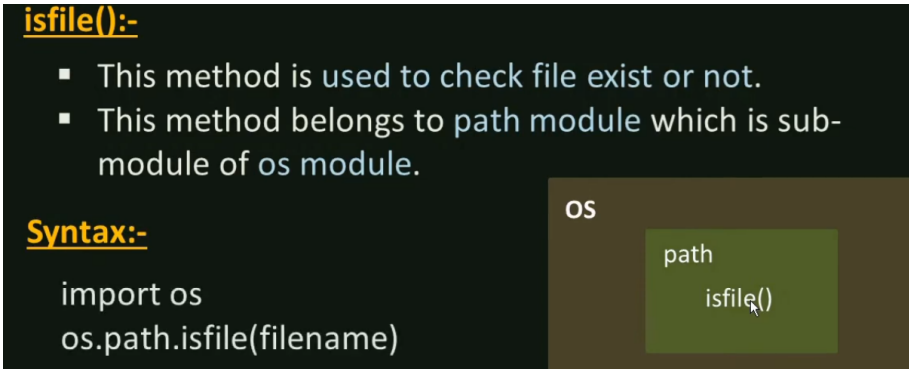
2. writable(). This method returns ‘True’ if the file is writable else it returns ‘False’

Example:

IPython: D:/Python Practice VS Code

```
In [17]: f=open("C:/Users/user/Desktop/The Matrix movie, insights.txt","r")

In [18]: f.readable()
Out[18]: True

In [19]: f.writable()
Out[19]: False

In [20]:
```

## Checking Whether a file exists or not:




Coding Example:

IPython: D:/Python Practice VS Code

```
In [28]: print(os.path.isfile("C:/Users/user/Desktop/The Matrix movie, insights.txt"))
True

In [29]: import os

In [30]: print(os.path.isfile("C:/Users/user/Desktop/The Matrix movie, insights.txt"))
True

In [31]:
```

## Ways of closing file:

Python doesn't guarantee that the close() function closes your file for sure. So, we will explore other ways of closing files:

- Using Exception Handling
- 'with' statement

### Using Exception Handling:

We can close a file in the 'finally' block. Because it is always executed whether an exception occurs or not.

Example:

```
try:
    f=open("C:/Users/user/Desktop/The Matrix movie, insights.txt",mode='r')
except Exception as e:
    print(e)
finally:
   f.close()
```

**Using 'with' statement:**

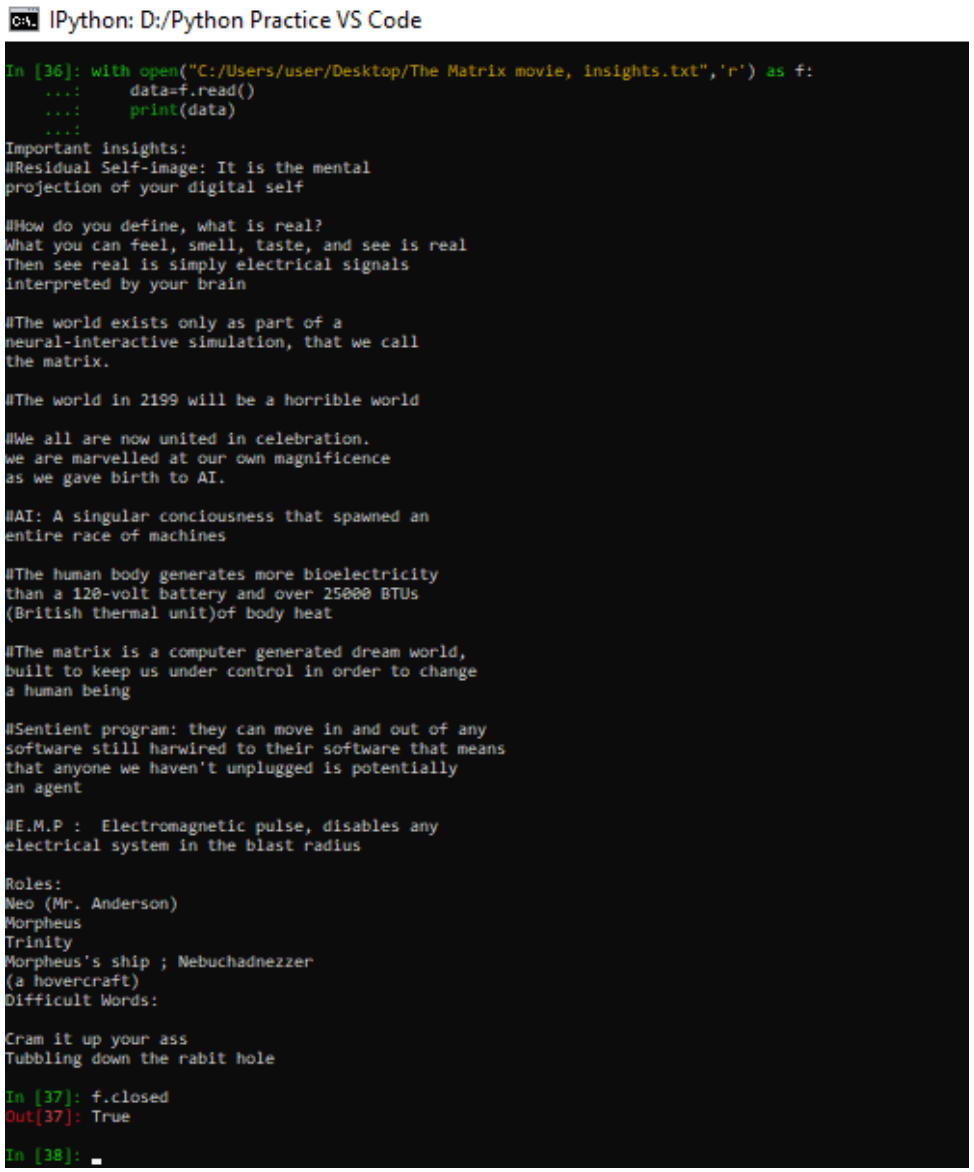
**'With'** statement will automatically close the file once the execution of 'with' block gets completed.

This is advantageous because it eliminates the need to explicitly call the close() method, reducing the risk of forgetting to close the file and ensuring proper resource management.

Code : `with open("C:/Users/user/Desktop/The Matrix movie, insights.txt",'r') as :`

```
...:     data=f.read()
```

```
...:     print(data)
```



## Mode of opening a file

Mode specifies the purpose of opening a file. We have two types of modes:

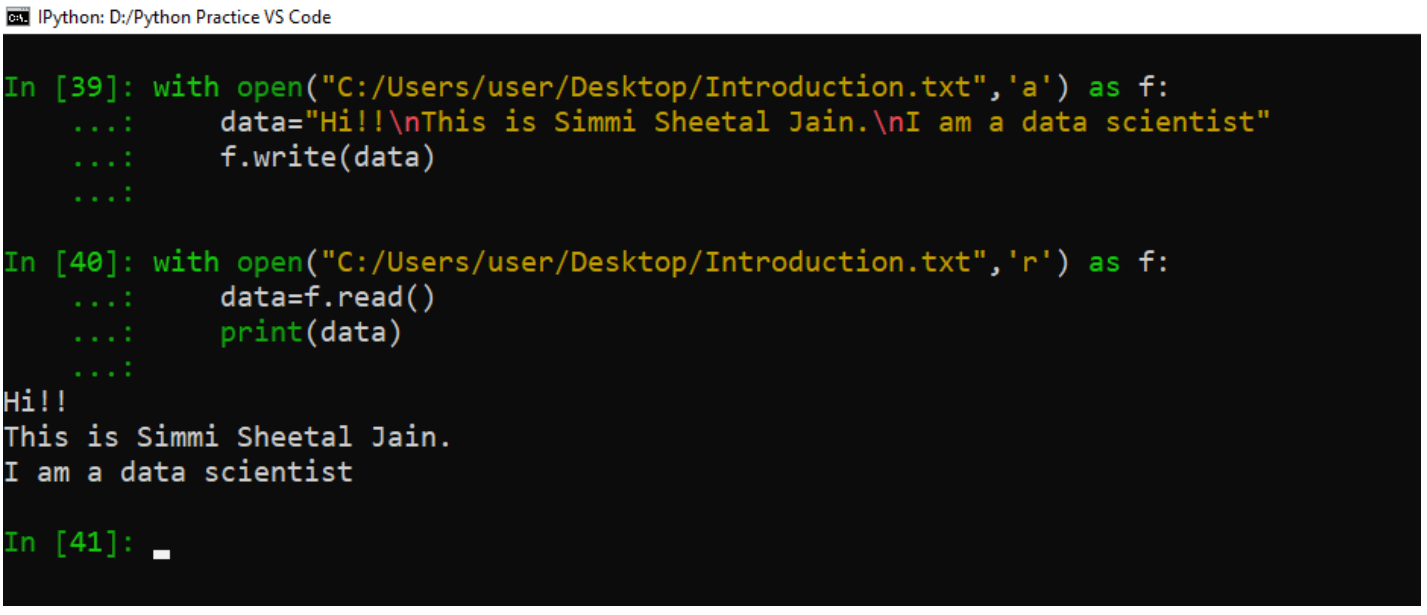
1. Text modes
2. Binary modes

**1. Text modes:** when we open a text file. Python first decodes the raw bytes using either a platform-dependent encoding or a specified encoding. Then, we perform operations such as read, write, append, etc.

The following are Text modes:

| | |
|---|---|
| r | Open file for reading only. If the file doesn't exist,it will show 'FileNotFoundError'.(default) |
| w | Open file for writing only. If the file doesn't exist,it will create a file. |
| a | Open for appending.It appends new data at end of file.If file does not exist,it creates a new file. |
| x | Open for exclusive creation for writing.The specified file must not be available.It creates a new file and then we write data into it.If file exists , it will give an error. |
| r+ | Open for reading and then writing. |
| w+ | Open for writing and then reading. |
| a+ | Open for appending and then writing. |

Examples:



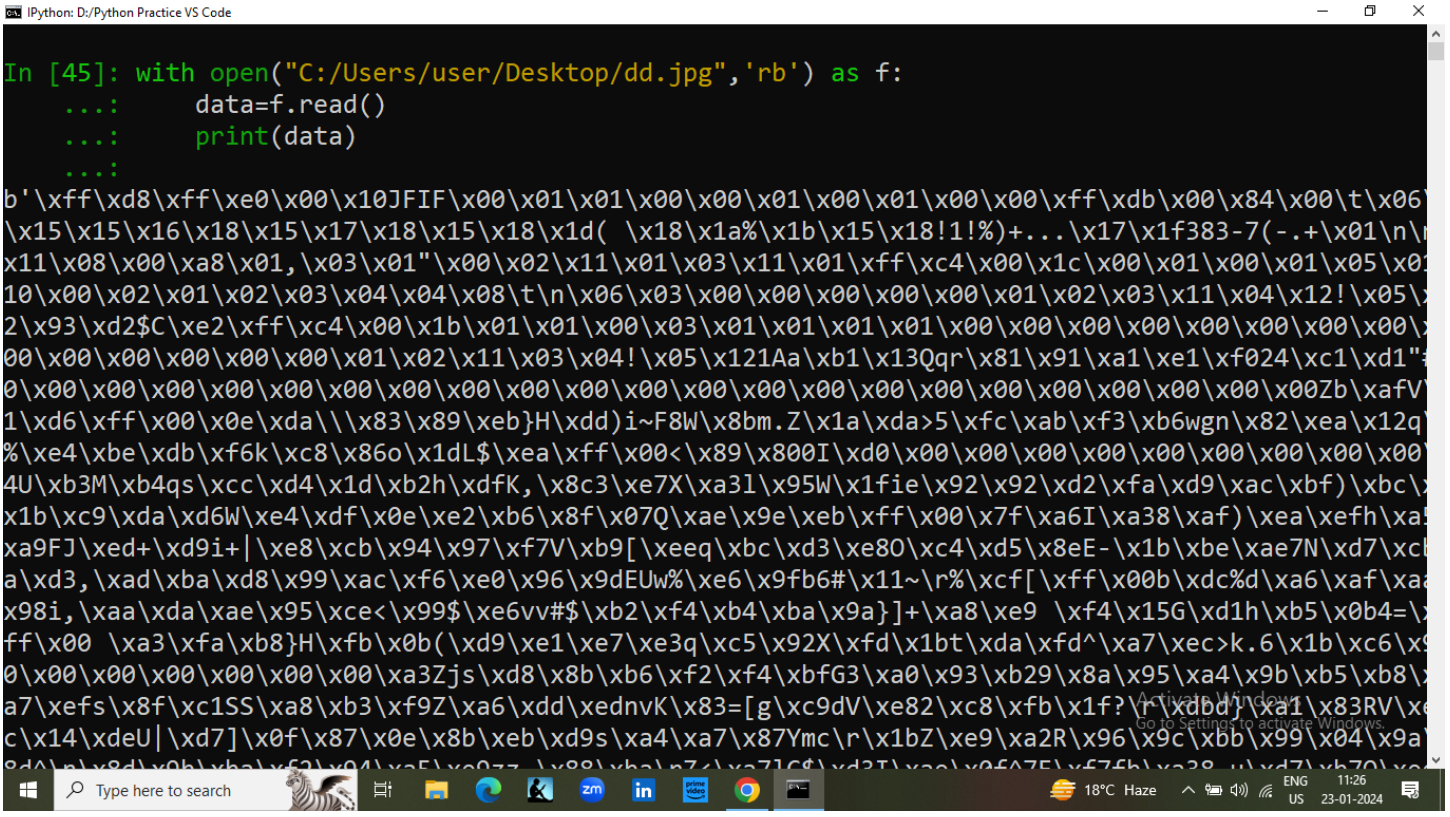
**2. Binary Modes:** When we open a file in binary mode, python uses the raw data without any decoding. The binary mode file reflects raw data directly in the Python file.

Python treats data of binary files as bytes. These modes are used with non-text files such as images, music, video, etc.

Following are common binary modes:

| | |
|---|---|
| rb | Open for reading in binary mode.(same as text 'b') |
| wb | Open for writing in binary mode.(same as text 'w') |
| ab | Open for appending.(same as text 'a'). |
| xb | Open for exclusive creation and writing.(same as 'x') |
| rb+ | Open for read and then write in binary. |
| wb+ | Open for writing and then reading in binary. |
| ab+ | Open for append and then read in binary |

Example:




## Reading Data From File

To read the content of a file, we have the following three functions:

- read

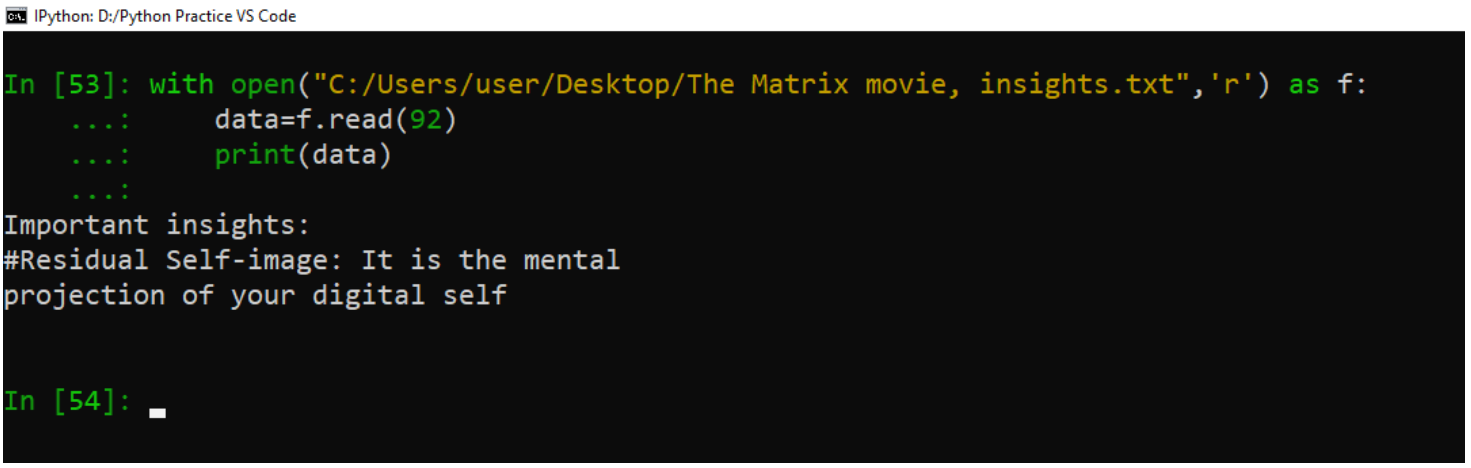
- readline()
- readlines()

**read():** It returns the data in string format if the mode is text mode, and returns bytes if the mode is binary mode.

Syntax: file_object.read(size)

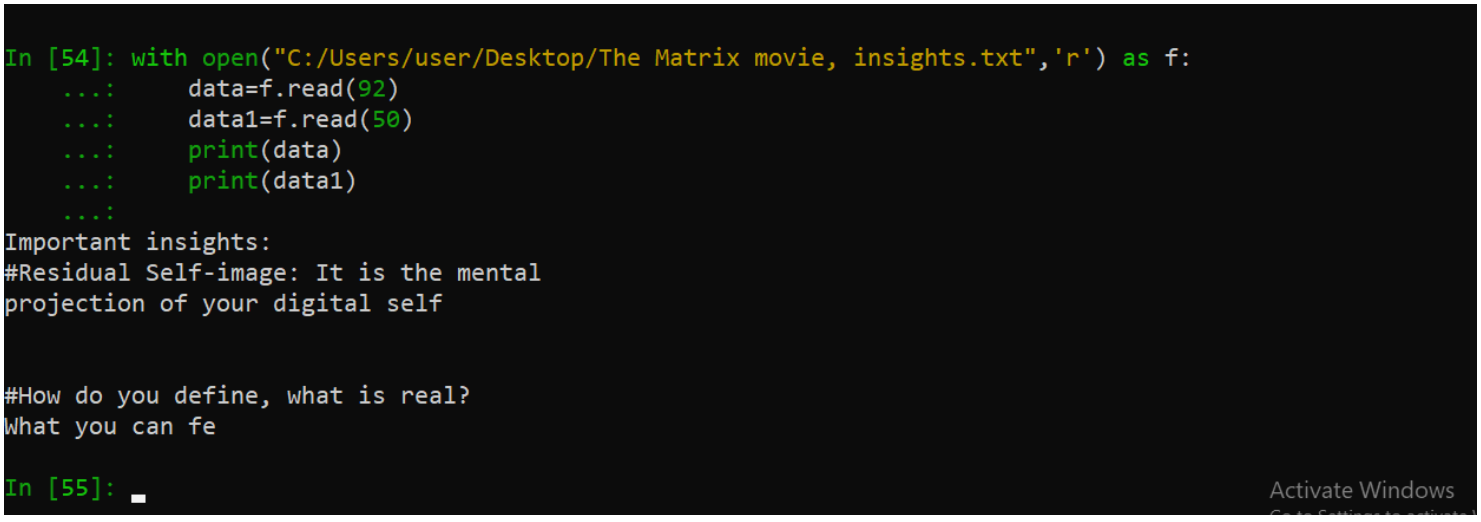
Here, size represents the number of characters to be read in read mode.

Code:




The file pointer in Python keeps track of the current position in the file, and each read or write operation advances the file pointer. If you read the file in the same program twice without closing it, the second read will indeed start from the current position of the file pointer.

Code:




**readline():** It reads a single line of characters from the file.

Syntax: file_object.readline(size)

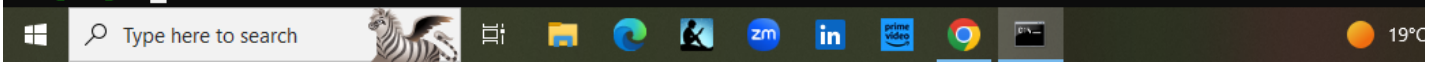
Here, size represents the number of characters to be read in read mode.

Code 1:

```
In [55]: with open("C:/Users/user/Desktop/The Matrix movie, insights.txt",'r') as f:
    ...:     data=f.readline(10)
    ...:     data1=f.readline(5)
    ...:     print(data)
    ...:     print(data1)
    ...:
Important
insig
```

Code 2:

```
In [56]: with open("C:/Users/user/Desktop/The Matrix movie, insights.txt",'r') as f:
    ...:     data=f.readline()
    ...:     data1=f.readline()
    ...:     print(data)
    ...:     print(data1)
    ...:
Important insights:

#Residual Self-image: It is the mental


In [57]:
```




**readlines():** It reads a list of lines from the file.

Syntax: file_object.readlines(size)

Here, size represents the number of characters to be read in read mode.

## Example:


After retrieving the list of lines, we can perform the operations on the list items:

Example:

```
In [62]: with open("C:/Users/user/Desktop/Intro.txt",'r') as f:
    ...:     data=f.readlines()
    ...:     for line in data:
    ...:         print(line)
    ...:
Hi!!

This is Simmi Sheetal Jain.

I am a data scientist

In [63]:
```

## Tell() and seek() method

- **Tell():** It tells the current position of the file pointer. The file pointer indicates the current position in the file. Initially, when a file is opened, the file pointer is set to position zero. The file pointer's position is conceptually similar to indexing in a string, representing the offset from the beginning of the file.

  Syntax: file_object.tell()

Example:

IPython: D:/Python Practice VS Code

```
In [68]: with open("C:/Users/user/Desktop/Intro.txt",'r') as f:
    ...:     position=f.tell()
    ...:     print(position)
    ...:     data=f.readline()
    ...:     print(data)
    ...:     position=f.tell()
    ...:     print(position)
    ...:     data=f.readline()
    ...:     print(data)
    ...:     position=f.tell()
    ...:     print(position)
    ...:
0
Hi!!

6
This is Simmi Sheetal Jain.

35

In [69]:
```

- **seek():** this method is used to change the position of the file pointer. Remember, file pointer always starts from 0.

  Syntax:

  file_object.seek(position)

Example:

IPython: D:/Python Practice VS Code

```
In [73]: with open("C:/Users/user/Desktop/The Matrix movie, insights.txt",'r') as f:
    ...:     f.seek(5)
    ...:     data=f.read(10)
    ...:     print(data)
    ...:     f.seek(0)
    ...:     data=f.read(100)
    ...:     print(data)
    ...:
tant insig
Important insights:
#Residual Self-image: It is the mental
projection of your digital self

#How do
```

## Find the Number of characters, words, and lines in a File

IPython: D:/Python Practice VS Code

```
In [91]: with open("C:/Users/user/Desktop/Intro.txt",'r') as f:
    ...:     print("The text inside file :\n_____________________________________\n ")
    ...:     data=f.read()
    ...:     for line in data:
    ...:         print(line,end="")
    ...:     f.seek(0)
    ...:     number_of_characters=0
    ...:     number_of_words=0
    ...:     number_of_lines=0
    ...:     for line in f:
    ...:         number_of_lines+=1
    ...:         line=line.strip("\n")
    ...:         number_of_characters+=len(line)
    ...:         list1=line.split()
    ...:         number_of_words+=len(list1)
    ...: print("\n_____________________\nNumber of lines are :", number_of_lines)
    ...: print("Number of words are :", number_of_words)
    ...: print("Number of characters are  : ",number_of_characters)
The text inside file :
_____________________________________

Hi!!
This is Simmi Sheetal Jain.
I am a data scientist
_____________________
Number of lines are : 3
Number of words are : 11
Number of characters are  :  52

In [92]:
```

## Writing data in a file

To write data in a file we have to open the file in w, a, x, etc. modes. Where ‘w’ is for write mode, ‘a’ is for append mode, and ‘x’ is for exclusive mode.

### ‘w’ mode:

- it is for writing only. It overwrites the data.

- In case, the file doesn't exist, it won't give an error rather will create a new file and write into it.
- The file pointer remains at the beginning of the file
- The file pointer exists at the beginning of the file.
- There are two main methods: write() and writelines()
- write() returns the number of characters to be printed in a file.

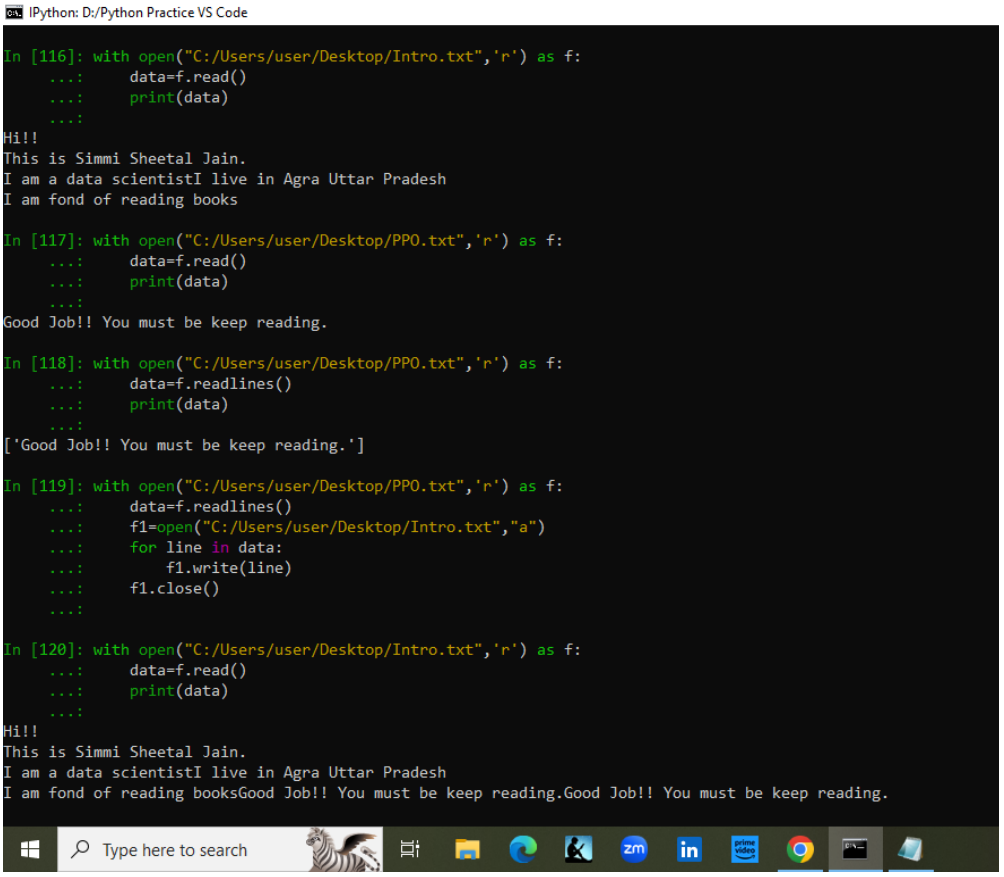
**'a' mode:**

- it is for appending something at the end of the text in the file.
- Code:

IPython: D:/Python Practice VS Code

```
In [97]: with open("C:/Users/user/Desktop/Intro.txt",'a') as f:
    ...:     data="\n"+input("Enter the new data string: ")
    ...:     f.write(data)
    ...:
Enter the new data string: I am fond of reading books

In [98]: with open("C:/Users/user/Desktop/Intro.txt",'r') as f:
    ...:     data=f.read()
    ...:     print(data)
    ...:
Hi!!
This is Simmi Sheetal Jain.
I am a data scientistI live in Agra Uttar Pradesh
I am fond of reading books

In [99]:
```

Copying Content of one file to another file



## writelines()

- It is used to write a list/set/tuple of strings on a file
- Syntax:
    - file_object.writelines(list/set/tuple of line)

Code:

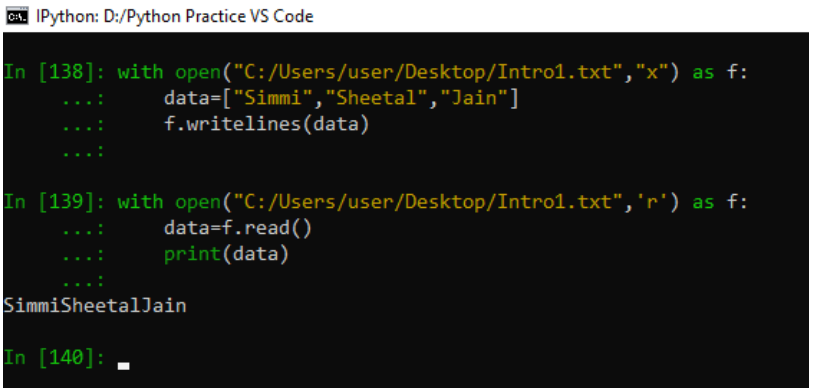
```
IPython: D:/Python Practice VS Code

In [122]: with open("C:/Users/user/Desktop/Intro.txt",'a') as f:
     ...:     data=["\nI am also fond of Solving Puzzles","\nI am also fond of travelling","\nI am also fond of writing blog and article"]
     ...:     f.writelines(data)
     ...:

In [123]: with open("C:/Users/user/Desktop/Intro.txt",'r') as f:
     ...:     data=f.read()
     ...:     print(data)
     ...:
Hi!!
This is Simmi Sheetal Jain.
I am a data scientistI live in Agra Uttar Pradesh
I am fond of reading booksGood Job!! You must be keep reading.Good Job!! You must be keep reading.
I am also fond of Solving Puzzles
I am also fond of travelling
I am also fond of writing blog and article

In [124]:
```

**'x' mode:** it doesn't write data only in a new file. If a file already exists, it will show an Error.

Code 1: When a File already exists:

```
IPython: D:/Python Practice VS Code

In [130]: f=open("C:/Users/user/Desktop/Intro.txt","r")

In [131]: data=f.read()

In [132]: print(data)
Hi!!
This is Simmi Sheetal Jain.
I am a data scientistI live in Agra Uttar Pradesh
I am fond of reading booksGood Job!! You must be keep reading.Good Job!! You must be keep reading.
I am also fond of Solving Puzzles
I am also fond of travelling
I am also fond of writing blog and article

In [133]: f=open("C:/Users/user/Desktop/Intro.txt","x")
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
Cell In[133], line 1
----> 1 f=open("C:/Users/user/Desktop/Intro.txt","x")

File ~\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:284, in _modified_open(file, *args, **kwargs)
    277 if file in {0, 1, 2}:
    278     raise ValueError(
    279         f"IPython won't let you open fd={file} by default "
    280         "as it is likely to crash IPython. If you know what you are doing, "
    281         "you can use builtins' open."
    282     )
--> 284 return io_open(file, *args, **kwargs)

FileExistsError: [Errno 17] File exists: 'C:/Users/user/Desktop/Intro.txt'

In [134]:
```

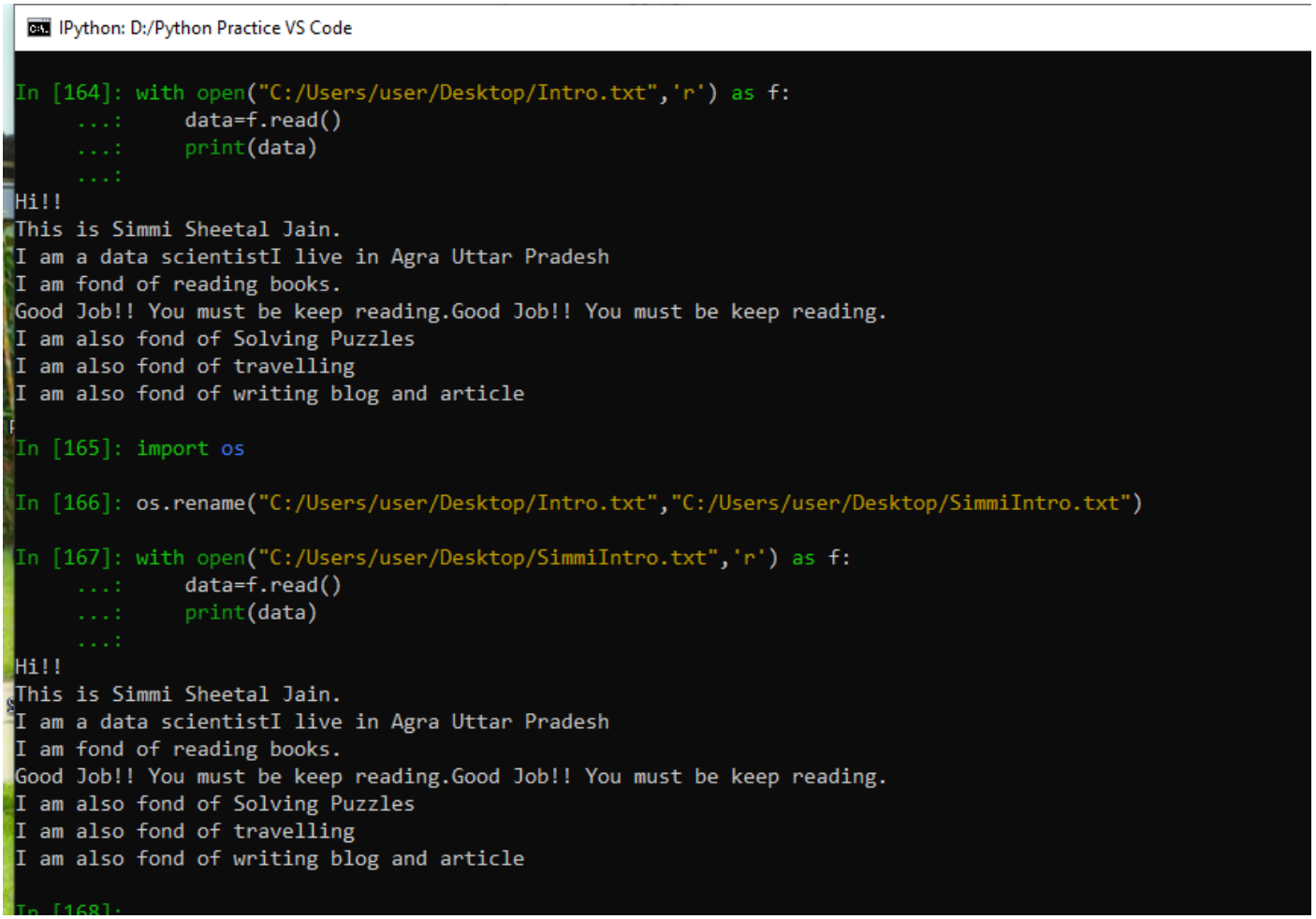
Code 2: When a new file is created:




## Renaming a file

We can use the os module to rename a file:

Here is how we can rename a file:




—----------The End—----------