

Introduction

Python programs accept the input, manipulate it & display the output. But output is available during execution of the programs & o/p is to be entered through the keyboard.

Variables used in programs have lifetime that last till the time, program is under execution. If we have to store data (i/p & o/p) permanently so that we can reuse it later, we use files.

Files

A file is a collection of data stored on a secondary storage device like hard disk.

File path - file that we use are used to store on a storage medium like hard disk. You have to give the file path to open that file.

There are 2 types of paths

- Relative Path - current working directory

Ex file1.txt

- Absolute Path - root & complete directory list

Ex - C:\student\documents\file1.txt

Types of files

Computer store every file as a collection of 0s & 1s i.e. Binary form. Every file is basically just a series of bytes stored one after the other.

Types of files

Text Files

Binary Files

(1) Text Files

- A text file can be understood as a sequence of characters consisting of alphabets, numbers & other special symbol.

Ex → .txt, .csv, .py etc

- When we open a file in text editor (eg. Notepad) we see several lines of text but internally it stored in sequence of bytes consisting of 0s & 1s.
- A text file consist human readable characters.

(2) Binary Files

- Binary files are also stored in terms of bytes (0s & 1s) but unlike text files, these bytes do not represent the ASCII value of characters.
- Binary files represent the actual content such as image, audio, video, compressed version of other files.
- These files are not human readable.
- If we try to open in text editor it will show some garbage value.

Opening and closing a Text Files

Opening File

To open a file in python, we use the `open()` function.

Syntax

file-object = `open(file-name [, access-module] [, buffering])`

file-object → This `open()` funⁿ returns a file-object, which is stored in a variable file-object. We can use this variable to transfer data to & from the file (read & write) by calling the funⁿ.

file-name → The file-name argument is a string value that contains the name of the file that you want to access. If the file does not exist, it will create a new file with assigned name.

access-mode → It determines the mode in which the file has to be opened.

• read • write • append etc.

• This is an optional parameter & the default file access mode is read (r)

buffering - If the buffering value is set to 0, no buffering takes place.

If value is set to 1, line buffering is performed while accessing file.

If value > 1, then buffering action performed with the indicated buffer size.

If value < 1, it will show some default behaviour.

File opening modes

<u>file mode</u>	<u>Description</u>	<u>File-Offset position</u>
<r>	opens the file in read-mode only	beginning of the file.
<rb>	opens the file in binary & read-mode only	beginning of the file.
<r+> <u>or</u> <+r>	opens the file in both read & write mode	beginning of the file.
<rb+> <u>or</u> <+rb>	open the file in binary read & binary write mode	beginning of the file.
<w>	open the file in write mode If file does not exist: creates a new file else: overwritten the existing file	beginning of the file
<wb>	open the file for writing in binary format. If file does not exist: creates a new file else: overwritten the existing file	beginning of the file.
<w+> <u>or</u> <+w>	open the file in writing & reading mode. If file does not exist: creates a new file else: overwritten the existing file	beginning of the file.
<wb+> <u>or</u> <+wb>	open the file for writing & reading in binary format. If file does not exist: creates a new file else: overwritten the existing file	beginning of the file

- <a> opens the file in append mode. end of the file.
If the file does not exist:
new file will be created
- <ab> open the file for append in binary format. end of the file.
If the file does not exist:
new file will be created
- <a+> or <+a> open the file for both appending & reading end of the file.
If the file does not exist:
new file will be created
- <ab+> or <+ab> open the file for both appending & reading in binary format end of the file.
If the file does not exist:
new file will be created

File object Attribute

The file-object has certain attributes that tells us basic information about the file, such as -

- <file.closed>
returns True if file is closed
- <file.mode>
returns the access mode in which the file was opened
- <file.name>
returns the name of the file
- <file.softspace>
returns False if space explicitly required with print,
True otherwise

Example

```
f = open("file.txt", "wb")
print(f.name)
print(f.closed)
print(f.mode)
print(f.softspace)
```

o/p
f.txt
False
wb
0

Closing File

- Once we are done with the read/write operations on a file, it is a good practice to close the file.
- Python provides a `close()` method to do so.
- While closing a file, the system frees the memory allocated to it.

syntax `file-object.close()`

Example -

```
# opens the file in read mode
f = open("file.txt", "r")
if f:
    print("file is opened successfully")
# closes the opened file
f.close()
```

after closing the file, we can't perform any operation. If any exception occurs while performing some operations in the file then the program terminates without closing the file.

- we can overcome this problem using `try & finally`.

```
try:
    f = open("file.txt")
finally:
    f.close()
```

once the file is closed using the `close()` method, any attempt to use the file-object will result in an error.

flush() method

The `flush()` method clears the internal buffer of the file.

Files are automatically flushed while closing them. A programmer can flush file before closing it by using `flush()` method. syntax - `file-object.flush()`

Example

(It does not require any parameters)

opening a file in read file

```
f = open("file.txt", "r")
```

clearing the i/p buffer

```
f.flush()
```

reading content

```
fData = f.read()
```

```
print(fData)
```

closing the file

```
f.close()
```

opening a file using with clause

We can also open a file using with clause.

syntax

with open (<file-name> , access-mode) as file-object :

- The advantage of using with clause is that any file opened using with clause is closed automatically once the control comes outside the with clause.
- In case the user forgets to close the file explicitly or if an exception occurs, the file is closed automatically.

```
with open ("myfile.txt", "r+") as f :
```

```
content = f.read()
```

Here we don't have to close the file explicitly using `close()` statement. Python will automatically close the file.

Reading from a text file

We can read the content of text file. Before reading a file, we must make sure that the file is opened in "r", "r+", "w+" or "a+" mode.

There are three ways to read the content of a file:

- (1) read() method
- (2) readline([n]) method
- (3) readlines() method

(1) The read() method

This method is used to read a specified numbers of bytes of data from a data file.

Syntax file-object.read(n)

The image contains two screenshots of a Jupyter Notebook interface. The top screenshot shows a code cell with a `with open("file.txt", "r") as f:` block containing `f.read(10)`. The output shows the first 10 characters of the file: "This is a text file.". Below it, another code cell shows `print(f.read())`, which outputs the remaining content of the file: "This is second line of text file. This is third line of text file. This is fourth line of text file. This is fifth line of text file.". The right pane shows the content of `file.txt` with five lines of text. The bottom screenshot shows a code cell with a `with open("file.txt", "r") as f:` block containing a `for line in f:` loop with `print(line)`. The output shows five lines, each being "This is a first line.". The right pane shows the content of `file.txt` with five lines, each being "This is a first line.". The status bar at the bottom indicates the file is at line 1, column 20, with 4 spaces, using CRLF line endings.

```
fileHandling.ipynb > with open("file.txt", "r") as f:
+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ... Python 3.12.0

if argument is passed.
markdown

with open("file.txt", "r") as f:
    f.read(10)
[24] ✓ 0.0s Python

if no argument and negative integer is passed
markdown

▶ with open("file.txt", "r") as f:
    print(f.read())
[26] ✓ 0.0s Python

...
This is a text file.
This is second line of text file.
This is third line of text file.
This is fourth line of text file.
This is fifth line of text file.

file.txt x
file.txt
1 This is a text file.
2 This is second line of text file.
3 This is third line of text file.
4 This is fourth line of text file.
5 This is fifth line of text file.
6

fileHandling.ipynb x
fileHandling.ipynb > looping over file
+ Code + Markdown | ▶ Run All | Restart ... Python 3.12.0

looping over file
markdown

with open("file.txt", "r") as f:
    for line in f:
        print(line)
[62] ✓ 0.0s Python

...
This is a first line.
This is a first line.
This is a first line.
This is a first line.
This is a first line.

file.txt x
file.txt
1 This is a first line.
2 This is a first line.
3 This is a first line.
4 This is a first line.
5 This is a first line.
6 This is a first line.

Ln 1, Col 20 Spaces: 4 CRLF Cell 1 of 2 Prettier 1514 26-10-2023
```


(2) readline([n]) method

This method will read the file line by line.

Example

```
fileHandling.ipynb
fileHandling.ipynb > with open("file.txt","r") as f:
+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Python 3.12.0

if argument is passed.
markdown

with open("file.txt","r") as f:
| print(f.readline(9))

[30] ✓ 0.0s Python
... This is a

if no argument and negative integer is passed
markdown

with open("file.txt","r") as f:
| print(f.readline(-6))

[31] ✓ 0.0s Python
... This is a text file.
```

```
file.txt
1 This is a text file.
2 This is second line of text file.
3 This is third line of text file.
4 This is fourth line of text file.
5 This is fifth line of text file.
6
```

(3) readlines() method

```
fileHandling.ipynb
fileHandling.ipynb > with open("file.txt","r") as f:
+ Code + Markdown | Run All | Restart | Clear All Outputs | Variables | Python 3.12.0

if argument is passed.
markdown

with open("file.txt","r") as f:
| print(f.readlines(9))

[32] ✓ 0.0s Python
... ['This is a text file.\n']
+ Code + Markdown

if no argument and negative integer is passed
markdown

with open("file.txt","r") as f:
| print(f.readlines())

[33] ✓ 0.0s Python
... ['This is a text file.\n', 'This is second line of text file.\n', 'This is third line of
```

```
file.txt
1 This is a text file.
2 This is second line of text file.
3 This is third line of text file.
4 This is fourth line of text file.
5 This is fifth line of text file.
6
```

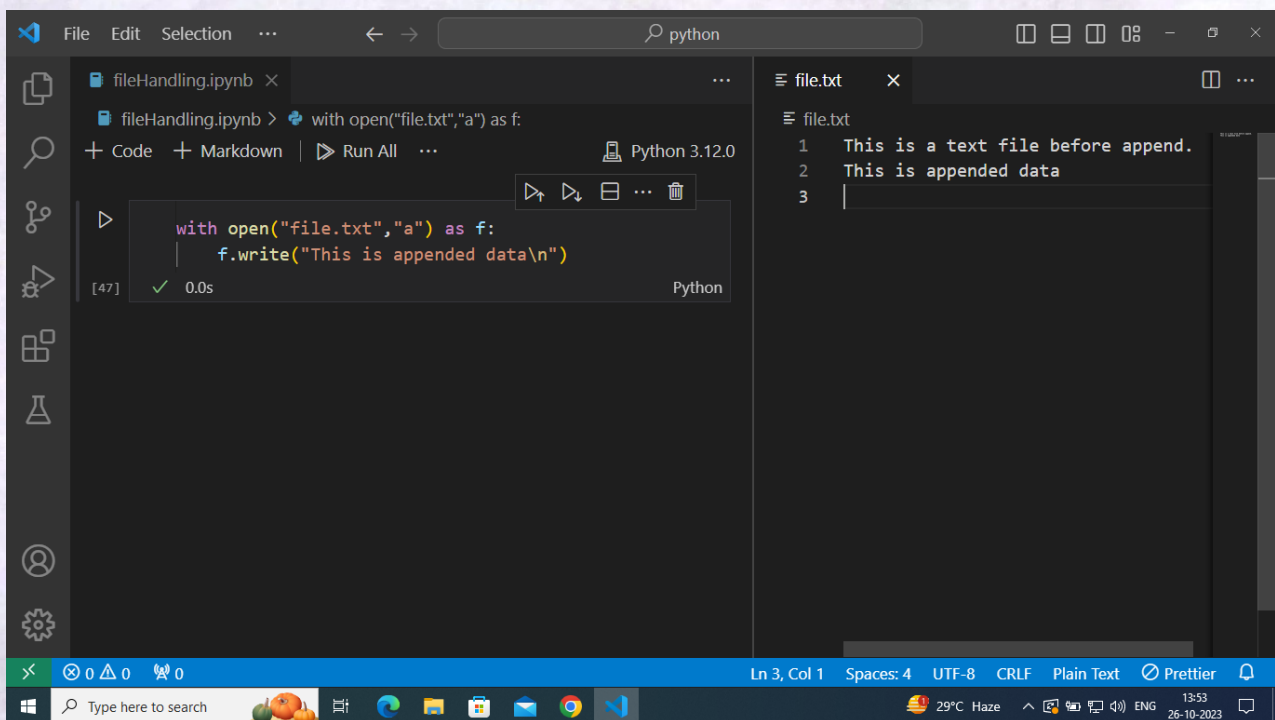
• lines in the file become members of a list, where each list element ends with a newline character ('\\n').

Writing to a text file

- For writing to a file, we first need to open it in write or append mode.
- If we open a file in write mode, previous data will be erased, file object will be positioned at the beginning of the file.
- If we open a file in append mode, new data will be added at the end of the previous data, file object is at the end of the file.
- There are 2 ways to write data in file-
 - `write()` - for writing a single string
 - `writelines()` - for writing a sequence of strings

(1) The write() method

open file in append mode



The screenshot shows a Jupyter Notebook window with a dark theme. The left sidebar contains icons for file explorer, search, and other notebook functions. The main area is divided into two panes. The left pane shows a code cell with the following Python code:

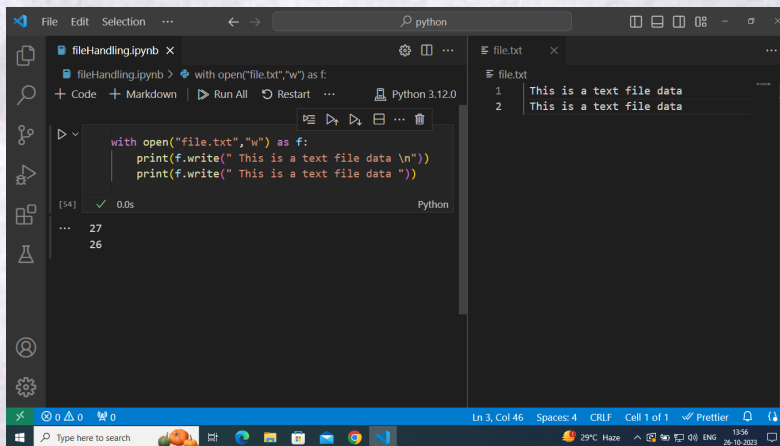
```
with open("file.txt", "a") as f:  
    f.write("This is appended data\n")
```

Below the code cell, the output is displayed: `[47] ✓ 0.0s Python`. The right pane shows the content of the file `file.txt`, which contains two lines of text:

```
1 This is a text file before append.  
2 This is appended data  
3
```

The bottom status bar of the Jupyter Notebook shows the current cursor position (Ln 3, Col 1), indentation (Spaces: 4), encoding (UTF-8), line endings (CRLF), text format (Plain Text), and a Prettier icon. The Windows taskbar at the very bottom shows the search bar, taskbar icons, and system tray information including temperature (29°C), weather (Haze), and date/time (13:53, 26-10-2023).

~~##~~ open file in write mode



The screenshot shows a Jupyter Notebook interface. The code cell contains the following Python code:

```
with open("file.txt", "w") as f:  
    print(f.write(" This is a text file data \n"))  
    print(f.write(" This is a text file data "))
```

The output cell shows the execution results:

```
[54] ✓ 0.0s Python  
... 27  
    26
```

The file.txt is open in a separate window, showing the content:

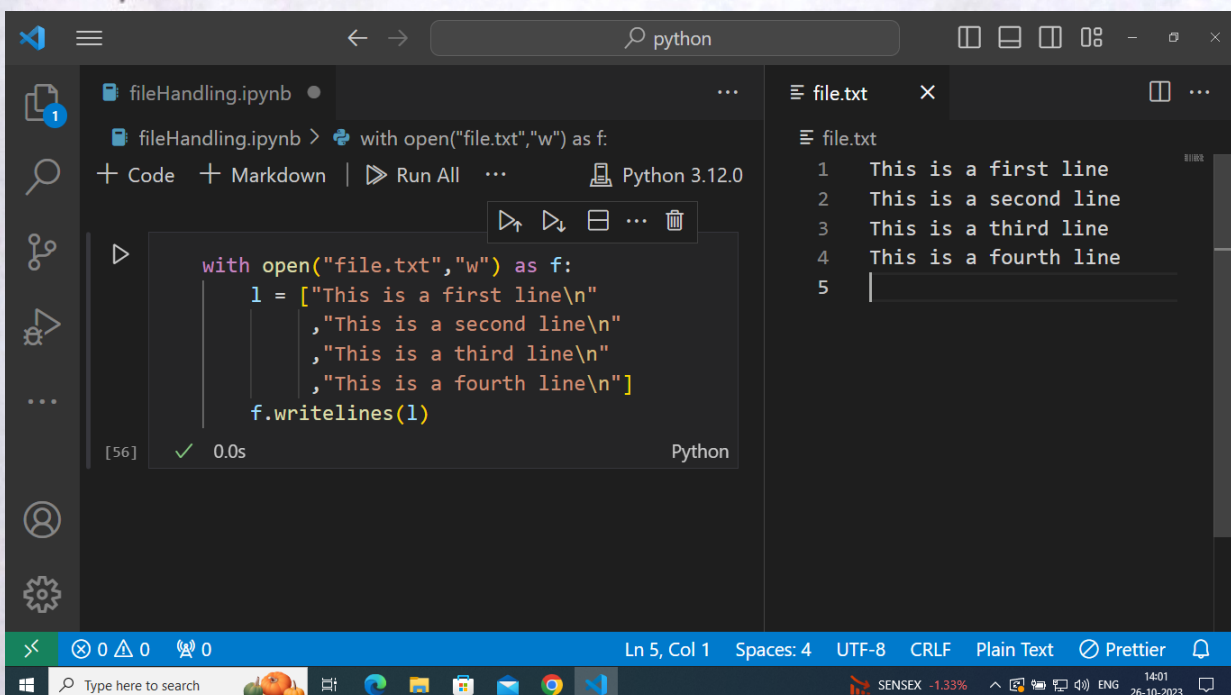
```
1 This is a text file data  
2 This is a text file data
```

Here, 27 & 26 are the length of string
'\n' → new line escape sequence
'\n' → treated as single character

Note

The `writel()` method writes data onto a buffer. When `close()` method is executed, the content from this buffer are moved to the file located in permanent storage. [This is done when we open a file without using 'with' clause.]

(2) writelines() method
write sequence of strings using writelines.
#open file in write mode



The screenshot shows a Jupyter Notebook interface. The code cell contains the following Python code:

```
with open("file.txt", "w") as f:  
    l = ["This is a first line\n",  
         "This is a second line\n",  
         "This is a third line\n",  
         "This is a fourth line\n"]  
    f.writelines(l)
```

The output cell shows the execution results:

```
[56] ✓ 0.0s Python
```

The file.txt is open in a separate window, showing the content:

```
1 This is a first line  
2 This is a second line  
3 This is a third line  
4 This is a fourth line  
5
```


open file in append mode

The screenshot shows a Jupyter Notebook interface. The left sidebar contains icons for file management, search, and settings. The main area is divided into two panes. The top pane, titled 'fileHandling.ipynb', shows a code cell with the following Python code:

```
with open("file.txt","a") as f:  
    l = ["This is a first line\n",  
        "This is a second line\n",  
        "This is a third line\n",  
        "This is a fourth line\n"]  
    f.writelines(l)
```

 Below the code cell, the output shows '[57] ✓ 0.0s Python'. The bottom pane, titled 'file.txt', displays the contents of the file:

```
1 This is a first line  
2 This is a second line  
3 This is a third line  
4 This is a fourth line  
5 This is a first line  
6 This is a second line  
7 This is a third line  
8 This is a fourth line  
9
```

 The status bar at the bottom indicates 'Ln 1, Col 22', 'Spaces: 4', 'UTF-8', 'CRLF', 'Plain Text', and 'Prettier'.

Program - Read the content from one file and copy it to another file.
OR
copy the content of the file.

The screenshot shows a Jupyter Notebook interface. The left sidebar contains icons for file management, search, and settings. The main area is divided into two panes. The top pane, titled 'fileHandling.ipynb', shows a code cell with the following Python code:

```
with open("file.txt","r") as f:  
    data = f.read()  
    print(data)
```

 Below the code cell, the output shows the contents of 'file.txt':

```
This is a first line  
This is a second line  
This is a third line
```

 The bottom pane, titled 'copy.txt', displays the contents of the file:

```
1 This is a first line  
2 This is a second line  
3 This is a third line  
4
```

 The status bar at the bottom indicates 'Ln 2, Col 18', 'Cell 4 of 4', and 'Prettier'.

FileEditSelectionViewGoRun...python

fileHandling.ipynb X

fileHandling.ipynb > with open("copy.txt","w") as f:

+ Code + Markdown | ▶ Run All ↺ Restart ≡ Clear All Outputs | Variables Outline ... Python 3.12.0

read the data from file.txt and copy into copy.txt

markdown

```
with open("file.txt","r") as f:
    data = f.read()
    print(data)
```

[59] Python

... This is a first line
This is a second line
This is a third line

Now we have data, our data is to write data into new file

markdown

▶ ▼

```
with open("copy.txt","w") as f:
    f.write(data)
```

[1] Python

file.txt X

file.txt

1 This is a first line
2 This is a second line
3 This is a third line
4

copy.txt X

copy.txt

1 This is a first line
2 This is a second line
3 This is a third line
4

Ln 2, Col 18 Cell 4 of 4 ✓ Prettier 14:12 26-10-2023

Type here to search

NIFTYMID100 -1.73%

ENG

14:12 26-10-2023