

Introduction

Python programs accept the input, manipulate it & display the output. But output is available during execution of the programs & I/O 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/O) 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

- Absolute Path - root & complete directory list

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

Types of files

Computer stores 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 (e.g. 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 File

Opening File

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

Syntax

```
file-object = open(file-name [,access-mode] [.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 values 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+> or <+r> | opens the file in both read & write mode | beginning of the file. |
| <rb+> or <+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+> or <+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+> or <+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.
 If the file does not exist:
 new file will be created end of the file.
- <ab> open the file for append in binary format.
 If the file does not exist:
 new file will be created end of the file.
- <a+> or <+a> open the file for both appending & reading
 If the file does not exist:
 new file will be created end of the file
- <ab+> or <+ab> open the file for both appending & reading in binary format
 If the file does not exist:
 new file will be created end of the file.

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)
```

| | |
|------------|---|
| <u>o/p</u> | |
| 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.

Example

```
# opening a file in read mode
```

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

```
# clearing the i/p buffer
```

```
f.flush()
```

```
# reading content
```

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

```
print(fData)
```

```
# closing the file
```

```
f.close()
```

(It does not require any parameters)

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", "rt", "w+" or "at" 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 screenshot shows a Jupyter Notebook interface with two code cells and their outputs. The first cell contains:

```
if argument is passed.
```

The output shows the result of running `f.read(10)`. The second cell contains:

```
if no argument and negative integer is passed
```

The output shows the result of running `f.read()`, which prints all five lines of the file.

The screenshot shows a VS Code interface with two code cells and their outputs. The first cell contains:

```
looping over file
```

The second cell contains:

```
with open("file.txt","r") as f:  
    for line in f:  
        print(line)
```

The output shows the result of running `f.read()` for each line in the file.

(2) readline([n]) method

This method will read the file line by line.

Example

The screenshot shows a Jupyter Notebook interface with two code cells and their outputs. The first cell contains code to read the first 9 lines of a file:if argument is passed.
with open("file.txt","r") as f:
| print(f.readline(9))
[30] ✓ 0.0s
... This is aThe output is a single line: "This is a". The second cell contains code to read the last 6 lines of a file:if no argument and negative integer is passed?
with open("file.txt","r") as f:
| print(f.readlines(-6))
[31] ✓ 0.0s
... This is a text file.The output is a single line: "This is a text file.". The status bar at the bottom shows "Ln 1, Col 32 Spaces: 4 CRLF Cell 2 of 4".

(3) readlines() method

The screenshot shows a Jupyter Notebook interface with two code cells and their outputs. The first cell contains code to read all lines of a file:if argument is passed.
with open("file.txt","r") as f:
| print(f.readlines(9))
[32] ✓ 0.0s
... ['This is a text file.\n']
[+ Code] [+ Markdown]The output is a list: ['This is a text file.\n']. The second cell contains code to read all lines of a file:if no argument and negative integer is passed?
with open("file.txt","r") as f:
| print(f.readlines(0))
[33] ✓ 0.0s
... ['This is a text file.\n', 'This is second line of text file.\n', 'This is third line ofThe output is a list: ['This is a text file.\n', 'This is second line of text file.\n', 'This is third line of'. The status bar at the bottom shows "Ln 2, Col 23 Spaces: 4 CRLF Cell 4 of 4".

- 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

A screenshot of a Jupyter Notebook interface. On the left, there is a sidebar with various icons. In the center, a code cell contains the following Python code:

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

The code cell has a green checkmark and a 0.0s execution time. To the right, there is a preview pane titled "file.txt" showing the contents of the file:

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

The bottom of the screen shows a Windows taskbar with icons for File Explorer, Edge, and other applications. The system tray shows the date and time as 26-10-2023.

#open file in write mode

A screenshot of a Jupyter Notebook interface. On the left, the code cell contains:

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

The code in the cell is:

```
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 "))

[54] ✓ 0.0s
```

The output cell shows the results of the print statements:

```
27
26
```

On the right, the file tab shows the contents of file.txt:

```
file.txt
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 `writelines()` 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

A screenshot of a Jupyter Notebook interface. On the left, the code cell contains:

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

The code in the cell is:

```
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)

[56] ✓ 0.0s
```

On the right, the file tab shows the contents of file.txt:

```
file.txt
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

A screenshot of a Jupyter Notebook interface. On the left, there's a sidebar with icons for file handling, search, and other notebook operations. The main area shows two files: 'fileHandling.ipynb' and 'file.txt'. In 'fileHandling.ipynb', the following Python code is written:

```
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)
```

The output cell shows the code has run successfully with a duration of 0.0s. To the right, the 'file.txt' file is shown with its contents:

```
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
```

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

A screenshot of a Jupyter Notebook interface. The sidebar includes file handling, search, and other notebook operations. The main area shows three files: 'fileHandling.ipynb', 'file.txt', and 'copy.txt'. In 'fileHandling.ipynb', the following Python code is written:

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

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

The output cell shows the data from 'file.txt':

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

Below, a markdown cell contains the text: "Now we have data, our data is to write data into new file". In the next code cell, the following Python code is written:

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

The 'file.txt' file is shown with its original content:

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

The 'copy.txt' file is shown with the copied content:

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

The screenshot shows a Jupyter Notebook interface with a dark theme. On the left, there's a sidebar with various icons. The top bar has a search field containing "python".

fileHandling.ipynb

Code cell:

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

Output:

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

Cell 1:

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

Cell 2:

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

Output:

```
[59] ... This is a first line  
This is a second line  
This is a third line
```

Cell 3:

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

Cell 4:

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

Output:

file.txt

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

copy.txt

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

Bottom status bar:

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