

# FILE HANDLING IN PYTHON

## A C A D V I E W

---

Until now in our python programs we have been saving data in variables and data structures like list, dictionaries, tuples etc.

You must have noticed one crucial thing, the data remains saved only for as long as our python program is running, in plain terms, data was saved only locally (RAM to be more precise),

Since RAM is a volatile memory, there is no way to access the data stored once the program has been terminated or it's execution has been over.

Programming languages uses RAM as the default memory allocation location during program execution

Fortunately, Python does provide a more permanent manner to store and access our data, known as **FILES**.

When we are working with Python, you don't need to import a library in order to read and write files. Its handled natively in the language.

A **File** is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk), which can be later used to access the data long after our python program has terminated.

In Python the file operations takes place in the following order.

1. Creating/Opening the file
2. Reading or writing contents from/to it (perform operation)
3. Closing the file

## 1. Opening a File

- Python has a built-in function – open() to open files
- open() returns a file object
- It accepts three arguments
  - File
  - Mode
  - Buffering

```
file object = open(file, mode, buffering)
```

The filename that we specify here should be the name of the file that exists within the working directory(The folder which has our program) of the python program, otherwise python will throw an error "No such file or directory"

File	the name of the file to open/create
mode	specifies the mode to open file in. Reading, writing, appending etc.
Buffering	If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size

```
f = open("GameOfThrones.txt") # file present in the working directory
f = open("C:/Users/Pooja/Desktop/GameOfThrones.txt") # Full file path
```

## 1.1 modes:

'r'	Open a file for reading (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
't'	Open the file in text mode. (Default)
'b'	Open the file in binary mode.
'+'	Open a file for updating (reading & writing)

The modes can be combined to create a custom desired mode

rb+ - To read & write the data in binary mode

```
f = open("myFile.txt")      # equivalent to 'r' or 'rt'
f = open("myFile.txt", 'w') # write in text mode
f = open("myFile.txt", 'rb+') # read and write in binary mode
```

## 2.1 Reading a File in Python

Python provides three flexible methods to read the data from the file object obtained so far. These are

- i) `read(size)`
- ii) `readline()`
- iii) `readlines()`

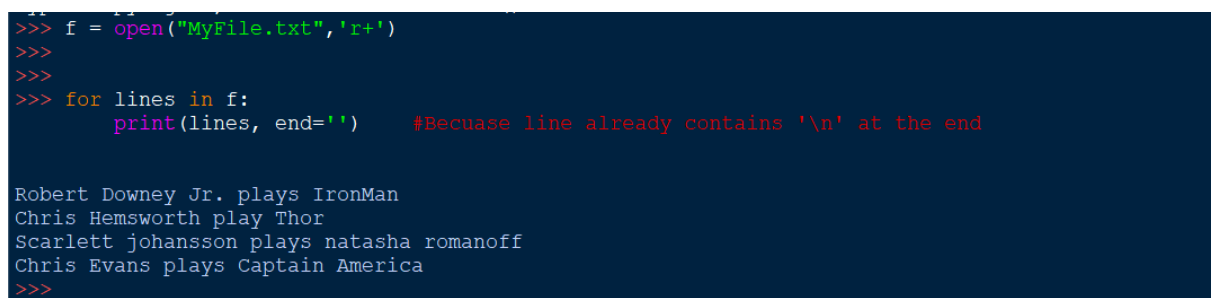
### 2.1.i) read(size)

the `read(size)` method is used to read in `size` number of data. If `size` parameter is not specified, it reads and returns up to the end of the file. For instance, in a text file number of data would mean number of characters to read.



```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
>>> # Opening and reading file contents
>>> f = open("MyFile.txt", 'r+')
>>> f.read(0) # Read first 0 data
''
>>> f.read(6) # Read next 6 data
'Robert'
>>> f.read(10) # Read the next 10 data
'Downey Jr'
>>> f.read() #Read entrie data until the end
'. plays IronMan\nChris Hemsworth play Thor\nScarlett johansson plays natasha romanoff\nChris Evans plays Captain America'
>>> f.read() #Read further data
File end reached, nothing to read!
>>>
```

Notice, the new lines are represented as `\n` characters. We can use a `for loop` and create a workaround by making use of the `end`



```
>>> f = open("MyFile.txt", 'r+')
>>>
>>>
>>> for lines in f:
>>>     print(lines, end='') #Becuase line already contains '\n' at the end

Robert Downey Jr. plays IronMan
Chris Hemsworth play Thor
Scarlett johansson plays natasha romanoff
Chris Evans plays Captain America
>>>
```

### 2.1.ii) readline()

The python `readline()` methods reads one line at a time from the file including the newline character (`\n`)

```

>>> file = open("MyFile.txt", 'r+')
>>> file.readline()
'Robert Downey Jr. plays IronMan\n'
>>>
>>> file.readline()      # read the next line
'Chris Hemsworth play Thor\n'
>>>
>>> file.readline()      # read the next line
'Scarlett johansson plays natasha romanoff\n'  Note the '\n' at the line end
>>>
>>> file.readline()
'Chris Evans plays Captain America'
>>>
>>> file.readline()
''
    ← File end reached, No more lines to read
>>>

```

### 2.1.iii) readlines()

The readlines() methods reads all the lines from the file one by one(sequentially) until the EndOfFile EOF is reached.

```

>>> file = open("MyFile.txt", 'r+')
>>>
>>> file.readlines()      #Using readlines() method
['Robert Downey Jr. plays IronMan\n', 'Chris Hemsworth play Thor\n', 'Scarlett johansson plays natasha romanoff\n', 'Chris Evans plays Captain America']
>>>

```

## 2.2 Writing into Files

- To write data into the file we use the python `write()` function
- It returns the number of characters written to the file
- CAUTION: To write data to a file, the file must be open in either
  - i) write `'w'`,
  - ii) append `'a'`,
  - iii) exclusive creation `'x'`
- using `'w'` to write the file will overwrite the file if it already exists, in other words, it will delete all the previous data.

### 2.2.i) Write mode 'w'

- This modes overwrites the file
- The original/old file data is completely deleted

- Only new data is present in the file after the operation

```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
>>> # Opening the file
>>> file = open("MyFile.txt", 'r')
>>> print(file.read())
Robert Downey Jr. plays IronMan
Chris Hemsworth play Thor
Scarlett johansson plays natasha romanoff
Chris Evans plays Captain AmericaJosh Brolin Plays THANOS
>>>
>>> # Writing new data to file
>>> file.close()
>>> file = open("MyFile.txt", 'w')
>>> file.write("Chriss Pratt Played StarLord")
28
>>> # Read the file
>>> file.close()
>>> file = open("MyFile.txt", 'r')
>>> print(file.read())
Chriss Pratt Played StarLord
>>>

```

Initial Data in the file

'w' for write mode

New Content written to file

File Existing data Deleted, New data written

## 2.2.ii) Append mode 'a'

- This modes adds the new contents to the end of the files.
- Original data is preserved
- New data is appended

```

Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> # opening the file
>>> file = open("MyFile.txt", 'r+')
>>> print( file.read() )
Robert Downey Jr. plays IronMan
Chris Hemsworth play Thor
Scarlett johansson plays natasha romanoff
Chris Evans plays Captain America
>>>
>>> file.close()
>>> file = open("MyFile.txt", 'a')
>>> file.write("Josh Brolin Plays THANOS")
24
>>> file.close()
>>> file = open("MyFile.txt", 'r+')
>>> print( file.read() )
Robert Downey Jr. plays IronMan
Chris Hemsworth play Thor
Scarlett johansson plays natasha romanoff
Chris Evans plays Captain AmericaJosh Brolin Plays THANOS
>>>

```

append mode to prevent overwriting the file

To append data into file

Add to the file

No of characters written to the file

New data appended.

## 2.2.iii) Exclusive Creation mode 'x'

- This opens files for exclusive creation.
- If the file already exists an error occurs.
- If the file doesn't exists already, a new file is created

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
>>> file = open("MyFile.txt", 'r')
>>> print(file.read())
Chriss Pratt Played StarLord
>>>
>>> file.close()
>>> file = open("MyFile.txt", 'x')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    file = open("MyFile.txt", 'x')
FileExistsError: [Errno 17] File exists: 'MyFile.txt'
>>>
>>> file = open("NewFile.txt", 'x')
>>>
```

exclusive creation mode

### 3. Closing the File

- When we have completed our file operation, we need to close our file
- The `close()` function is used to close the file
- On calling the `close()` function, the resources that were tied to file will be freed up.
- One must not rely on garbage cleaner to close the file and explicitly do so.

```
file.close()
```

### 4. File Read positions

#### 3.1) `tell()`

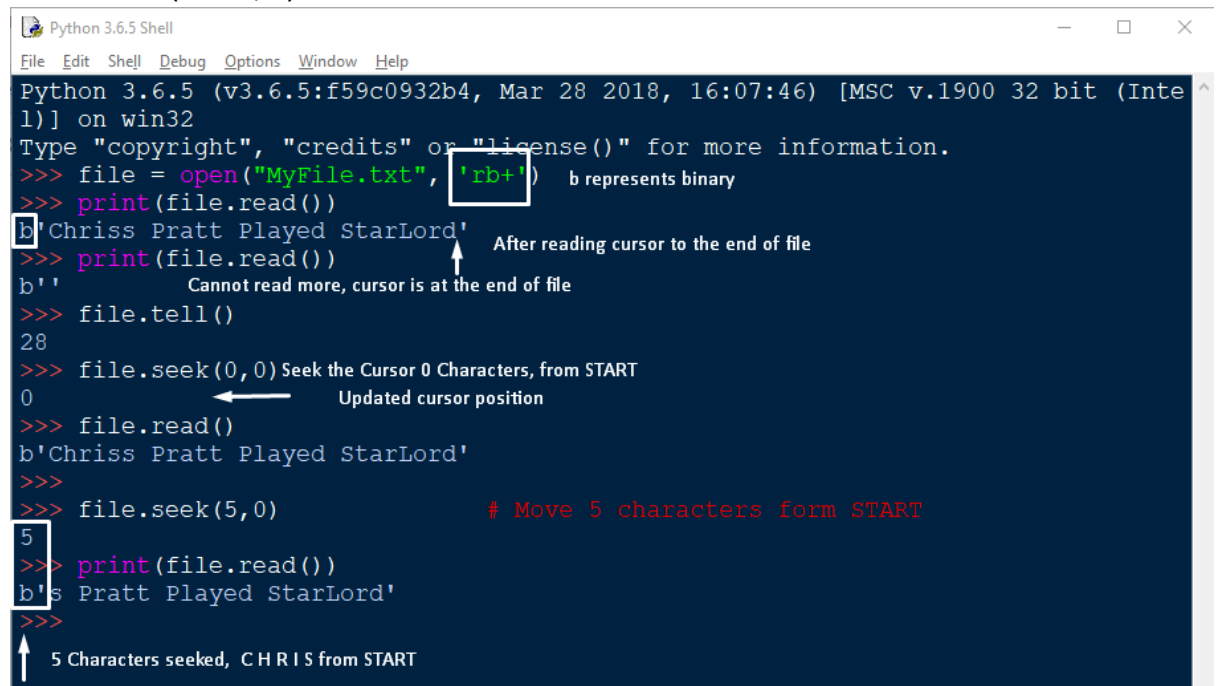
- This method returns the current location in the file

```
Python 3.6.5 Shell
File Edit Shell Debug Options Window Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
>>> file = open("MyFile.txt", 'r+')
>>> file.tell()
0
>>> print(file.read(5)) # Reading first 5 chars
Chris
>>> file.tell()
5
>>> print(file.read()) # Reading the remaining file
s Pratt Played StarLord
>>>
>>> file.tell() # file Position
28
>>> file.read(1)
' '
>>> file.tell()
28
>>>
```

### 3.2) seek(offset, from=SEEK\_SET)

- The seek function changes the cursor position in the file.
- It seeks backwards or forwards
- The offset determines the no. of characters to move say, int value
- The form specifies the seek operation from 3 position
  - 0 – Start of the file
  - 1 – Current Position
  - 2 – End of the file

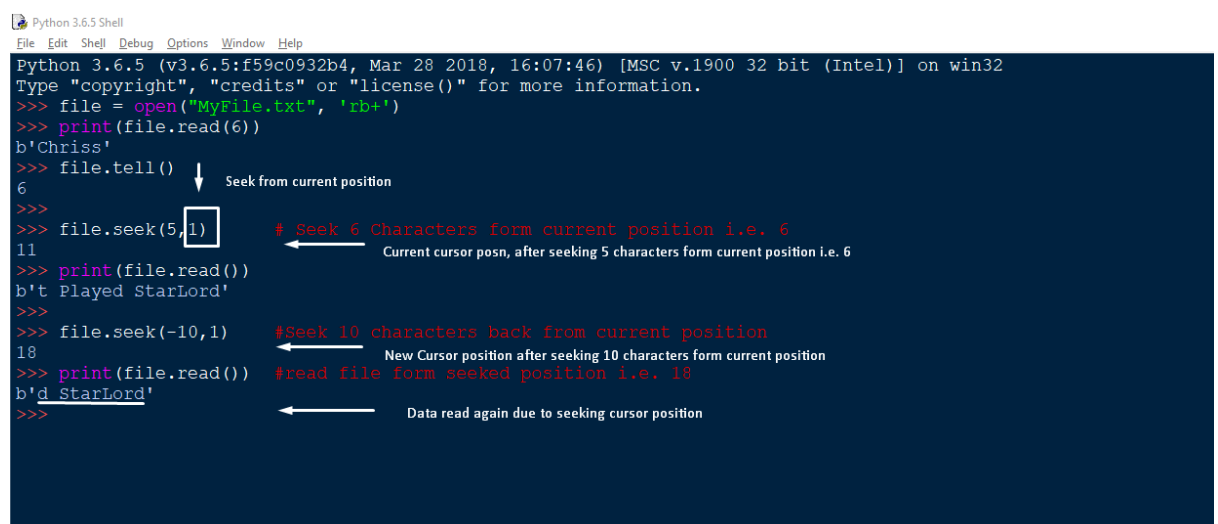
From start: (offset, 0)



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> file = open("MyFile.txt", 'rb+')  b represents binary
>>> print(file.read())
b'Chriss Pratt Played StarLord'
>>> print(file.read())
b''
      Cannot read more, cursor is at the end of file
>>> file.tell()
28
>>> file.seek(0,0) Seek the Cursor 0 Characters, from START
0
      Updated cursor position
>>> file.read()
b'Chriss Pratt Played StarLord'
>>>
>>> file.seek(5,0) # Move 5 characters form START
5
>>> print(file.read())
b's Pratt Played StarLord'
>>>
```

↑ 5 Characters seeked, C H R I S from START

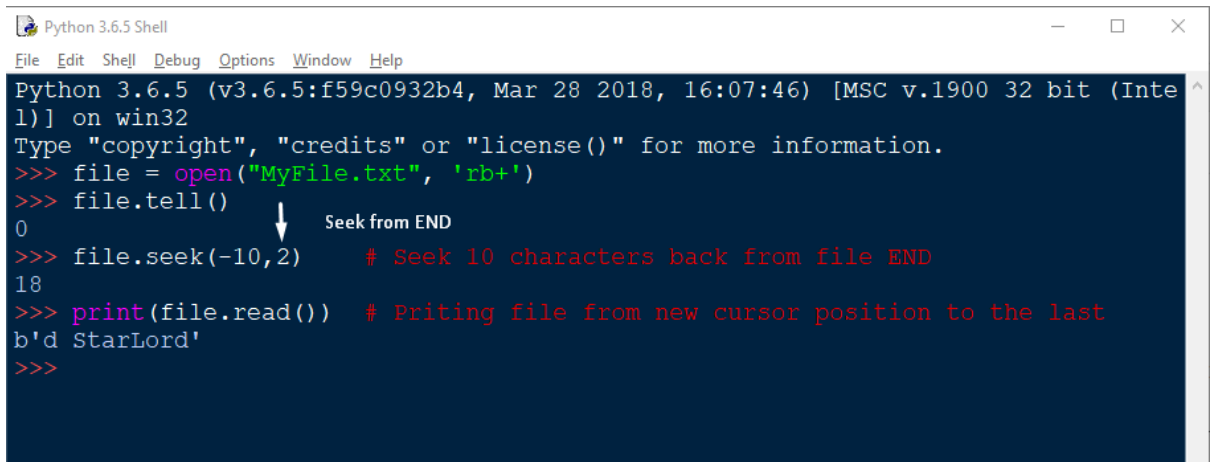
Form Current: seek(offset, 1)



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> file = open("MyFile.txt", 'rb+')
>>> print(file.read(6))
b'Chriss'
>>> file.tell()
6
      Seek from current position
>>>
>>> file.seek(5,1) # Seek 5 Characters form current position i.e. 6
11
      Current cursor posn, after seeking 5 characters form current position i.e. 6
>>> print(file.read())
b't Played StarLord'
>>>
>>> file.seek(-10,1) #Seek 10 characters back from current position
18
      New Cursor position after seeking 10 characters form current position
>>> print(file.read()) #read file form seeked position i.e. 18
b'd StarLord'
>>>
```

← Data read again due to seeking cursor position

From End: seek(offset, 2)

A screenshot of a Python 3.6.5 Shell window. The window has a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area shows a Python prompt with the following code and output:

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> file = open("MyFile.txt", 'rb+')
>>> file.tell()
0
>>> file.seek(-10,2) # Seek 10 characters back from file END
18
>>> print(file.read()) # Printing file from new cursor position to the last
b'd StarLord'
>>>
```

An arrow points from the text 'Seek from END' to the second parameter '2' in the seek function call.

## 5.Opening the files using ‘with’ statements

You can also work with file objects using the with statement. It is designed to provide much cleaner syntax and exceptions handling when you are working with code. That explains why it's good practice to use the with statement where applicable.

One bonus of using this method is that any files opened will be closed automatically after you are done. This leaves less to worry about during cleanup.

To use the with statement to open a file:

```
with open("MyFile.txt") as file:
    for lines in file:
        print(lines, end=' ')
```

The file object f is only valid inside the with block and the file closed automatically when we come out of the with block.

## 6.Renaming & Deleting the files

While you were using the read/write functions, you may also need to rename/delete a file in Python. So, there comes a os module in Python which brings the support of file rename/delete operations



## 6.1) Rename:

- To rename a file we need to import the `os module` and use the rename method provided in it

```
rename(cur_file_name, New_file_name)
```

- The `rename()` methods expects two arguments, the current filename and the new filename

Example:

```
import os
# Rename MyFile.txt → One.txt
os.rename("MyFile.txt", "One.txt")
```

## 6.2) Remove:

- In Python the os module provides the method `remove()`, that can be used to delete files in python
- The function expects only a single argument, the filename, to be deleted.

```
remove(filename)
```

example:

```
import os
# To delete MyFiles.txt
os.remove("MyFiles.txt")
```

## 7. Python File methods

<code>close()</code>	Close an open file. It has no effect if the file is already closed.
<code>detach()</code>	Separate the underlying binary buffer from the TextIOBase and return it
<code>fileno()</code>	Return an integer number (file descriptor) of the file
<code>flush()</code>	Flush the write buffer of the file stream.

isatty()	Return True if the file stream is interactive.
read(n)	Read atmost n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resize the file stream to size bytes. If size is not specified, resize to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Write string s to the file and return the number of characters written.
writelines(lines)	Write a list of lines to the file.

Function	Description
<code>&lt;file.close()&gt;</code>	Close the file. You need to reopen it for further access.
<code>&lt;file.flush()&gt;</code>	Flush the internal buffer. It's same as the <code>&lt;stdio&gt;</code> 's <code>&lt;fflush()&gt;</code> function.
<code>&lt;file.fileno()&gt;</code>	Returns an integer file descriptor.
<code>&lt;file.isatty()&gt;</code>	It returns true if file has a <code>&lt;tty&gt;</code> attached to it.
<code>&lt;file.next()&gt;</code>	Returns the next line from the last offset.
<code>&lt;file.read(size)&gt;</code>	Reads the given no. of bytes. It may read less if EOF is hit.
<code>&lt;file.readline(size)&gt;</code>	It'll read an entire line (trailing with a new line char) from the file.
<code>&lt;file.readlines(size_hint)&gt;</code>	It calls the <code>&lt;readline()&gt;</code> to read until EOF. It returns a list of lines read from the file. If you pass <code>&lt;size_hint&gt;</code> , then it reads lines equalling the <code>&lt;size_hint&gt;</code> bytes.
<code>&lt;file.seek(offset[, from])&gt;</code>	Sets the file's current position.
<code>&lt;file.tell()&gt;</code>	Returns the file's current position.
<code>&lt;file.truncate(size)&gt;</code>	Truncates the file's size. If the optional size argument is present, the file is truncated to (at most) that size.
<code>&lt;file.write(string)&gt;</code>	It writes a string to the file. And it doesn't return any value.
<code>&lt;file.writelines(sequence)&gt;</code>	Writes a sequence of strings to the file. The sequence is possibly an iterable object producing strings, typically a list of strings.