For instance, if a file **Master.txt** is opened *via* file-handle ***outfile***, it may be closed by following statement :

```
outfile.close( )
```

The close( ) must be used with filehandle

Please remember, **open( )** is a built-in function (used stand- alone) while **close( )** is a method used with file-handle object.

Info Box 5.1

Why should you close files in your program ?

You know that you can close an open file using the close( ) function. What if you don't use close( ) function on the file handle? Well, in that case Python will eventually close your file anyway. But then why is it recommended every time that one should close the open file using close( ) ?

The reasons for this are :

(*i*) Closing files using close() is a good programming practice.

(*ii*) Sometimes the data written onto files using write functions is held in memory until the next write or the file is closed. So for the last write operation, the data may remain in memory until close() is used. This may lead to data loss sometimes.

(*iii*) Sometimes there are operating system restrictions on number of open files at any time. Thus, you must close your file when its work is done.

(*iv*) Some operating systems treat open files as locked and private. When a file which is not closed and no longer in use, it leads to unnecessary blockage of memory.

For all the above reasons, it is advised that files should be closed in the programs by explicitly using close( ) function.

.4 WORKING WITH TEXT FILES

Python provides many functions for reading and writing the open files. In this section, we are going to explore these functions. Most common file reading and writing functions are being discussed in coming lines.

.4.1 Reading from Text Files

Python provides mainly *three* types of *read functions* to read from a data file. But before you can read from a file, the file must be opened and linked via a ***file-object*** or ***file handle***. Most common file reading functions of Python are listed below in Table 5.2.

Table 5.2 *Python data files — reading writing functions*

S.No.	Method	Syntax	Description

1.	read( )	`<filehandle>.read([n])`	reads at most *n* bytes ; if no *n* is specified, reads the entire file.
			Returns the read bytes in the form of a *string*.
			`>>> file1 = open("E:\\mydata\\info.txt")`
			`>>> readInfo = file1.read(15)`
			`>>> print(readInfo)`
			`It's time to wo`
			15 bytes read
			`>>> type(readInfo)`
			`str`
			Bytes read into string type

S.No.	Method	Syntax	Description

2.	readline( )	<filehandle>.readline([n])	reads a line of input ; if *n* is specified reads at most *n* bytes.
			Returns the read bytes in the form of a *string* ending with \n(line) character or returns a blank string if no more bytes are left for reading in the file.
			`>>> file1 = open("E:\\mydata\\info.txt")`
			`>>> readInfo = file1.readline()`
			`>>> print(readInfo)`
			`It's time to work with files.`
			1 line read

3.	readlines( )	<filehandle>.readlines()	reads all lines and returns them in a *list*
			`>>> file1 = open("E:\\mydata\\info.txt")`
			`>>> readInfo = file1.readline()`
			`>>> print(readInfo)`
			`["It's time to work with files.\n", 'Files offer and ease and power to store your work/data/information for later use.\n', 'simply create a file and store(write) in it .\n', 'Or open an existing file and read from it.\n']`
			All lines read
			`>>> type(readInfo)`
			`List`
			Read into list type

The <filehandle> in above syntaxes is the file-object holding open file's reference.

Let us consider some examples now. For the examples and explanations below, we are using a file namely *poem.txt* storing the content shown in Fig. 5.2.

WHY ?

We work, we try to be better
We work with full zest
But, why is that we just don't know any letter.

We still give our best.
We have to steal,
But, why is that we still don't get a meal.

We don't get luxury,
We don't get childhood,
But we still work,
Not for us, but for all the others.

Why is it that some kids wear shoes, BUT we make them ?

by Mythili, class 5

Figure 5.2 Contents of a sample file *poem.txt*.

Code Snippet 1 *Reading a file's first 30 bytes and printing it*

```
myfile = open(r'E:\poem.txt', "r")
str = myfile.read( 30 )
print(str)
```

File-object created → `myfile`

File-object being used → `myfile`

See the number of bytes to be read specified as argument of read( )

When we specify number of bytes with read( ), Python will read only the specified number of bytes from the file. The next read( ) will start reading onwards from the last position read. *Code snippet 2* illustrates this fact.

The output produced by above code is :

```
>>>
WHY?
We work, we try
```

TIP

You may combine the file opening with the file-object's function if you need to perform just a single task with the open file.

If need to perform just a single task with the open file, then you may combine the two steps of opening the file and performing the task, *e.g.*, following statement :

```
>>> file(r"E :\poem.txt", "r").read(30)
```

First function will open the file and the second function will perform with the result of first function, i.e., the reference of open file

would give the same result as that of above code.

For example, the following code will return the first line of file *poem.txt* :

```
open("poem.txt", "r").readline()
```

Code Snippet 2 *Reading n bytes and then reading some more bytes from the last position read*

```
myfile = open(r'E:\poem.txt', 'r')
str = myfile.read( 30 )
print(str)
str2 = myfile.read( 50 )
print(str2)
myfile.close()
```

reading 30 bytes

reading next 50 bytes

To see reading from file in action

Scan QR Code

The output produced by above code is :

```
>>>
WHY?
We work, we try
 to be better
We work with full zest
But, why is t
```

*Output by first print statement, i.e., **print(str)***

***print** has entered a new line after its output i.e., here*

*Output by second print statement, i.e., **print(str2)***

Snippet 5 *Reading a complete file - line by line*

```
myfile = open(r'E:\poem.txt', "r")
str = " "                    #initially storing a space (any non-None value)
while str :
    str = myfile.readline( )
    print(str, end = ' ')
myfile.close( )
```

The output produced by the above code will print the entire content of file poem.txt

```
>>>
    WHY?

We work, we try to be better
We work with full zest
But, why is that we just don't know any letter.

We still give our best.
We have to steal,
But, why is that we still don't get a meal.

We don't get luxury,
We don't get childhood,
But we still work,
Not for us, but for all the others.

Why is it that some kids wear shoes, BUT we make them ?

by      Mythili, class 5
```

The *readline( )* function reads the leading and trailing spaces (if any) along with trailing newline character('\n') also while reading the line. You can remove these leading and trailing white spaces (spaces or tabs or newlines) using *strip( )* (without any argument) as explained below. Recall that *strip( )* without any argument removes leading and trailing whitespaces.

There is another way of printing a file line by line. This is a simpler way where after opening file you can browse through the file using its file handle line by line by writing following code

```
<filehandle> = open(<filename>, [<any read mode>])
for <var> in <filehandle> :
    print(<var>)
```

For instance, for the above given file *poem.txt*, if you write following code, it will print the entire file line by line :

```
myfile = open(r'E:\poem.txt', "r")
for line in myfile :
    print(line)
```

The output produced by above code is just the same as the output produced by above program. The reason behind this output is that when you iterate over a file-handle using a for loop, the

Now that you are familiar with these reading functions' working, let us write some

Program 5.1 Write a program to display the size of a file in bytes.

```
myfile = open(r'E:\poem.txt', "r")
str = myfile.read()
size = len(str)
print("Size of the given file poem.txt is")
print(size, "bytes")
```

Output

```
>>>
Size of the given file
387 bytes
```

Program 5.2 Write a program to display the number of lines in the file.

```
myfile = open(r'E:\poem.txt', "r")
s = myfile.readlines()
linecount = len(s)
print("Number of lines in poem.txt is", linecount)
myfile.close()
```

Output

```
>>>
Number of lines in poem.tx
```