Chapter 2

File Handling in Python

There are many ways of trying to understand programs. People often rely too much on one way, which is called "debugging" and consists of running a partly-understood program to see if it does what you expected. Another way, which ML advocates, is to install some means of understanding in the very programs themselves.

— *Robin Milner*

12130CH02

In this *Chapter*

2.1 Introduction to Files

We have so far created programs in Python that accept the input, manipulate it and display the output. But that output is available only during execution of the program and input is to be entered through the keyboard. This is because the variables used in a program have a lifetime that lasts till the time the program is under execution. What if we want to store the data that were input as well as the generated output permanently so that we can reuse it later? Usually, organisations would want to permanently store information about employees, inventory, sales, etc. to avoid repetitive tasks of entering the same data. Hence, data are stored permanently on secondary storage devices for reusability. We store Python programs written in script mode with a .py extension. Each program is stored on the secondary device as a file. Likewise, the data entered, and the output can be stored permanently into a file.

So, what is a file? A file is a named location on a secondary storage media where data are permanently stored for later access.

2.2. TYPES OF FILES

Computers store every file as a collection of 0s and 1s i.e., in binary form. Therefore, every file is basically just a series of bytes stored one after the other. There are mainly two types of data files — text file and binary file. A text file consists of human readable characters, which can be opened by any text editor. On the other hand, binary files are made up of non-human readable characters and symbols, which require specific programs to access its contents.

2.2.1 Text file

A text file can be understood as a sequence of characters consisting of alphabets, numbers and other special symbols. Files with extensions like .txt, .py, .csv, etc. are some examples of text files. When we open a text file using a text editor (e.g., Notepad), we see several lines of text. However, the file contents are not stored in such a way internally. Rather, they are stored in sequence of bytes consisting of 0s and 1s. In ASCII, UNICODE or any other encoding scheme, the value of each character of the text file is stored as bytes. So, while opening a text file, the text editor translates each ASCII value and shows us the equivalent character that is readable by the human being. For example, the ASCII value 65 (binary equivalent 1000001) will be displayed by a text editor as the letter 'A' since the number 65 in ASCII character set represents 'A'.

Each line of a text file is terminated by a special character, called the End of Line (EOL). For example, the default EOL character in Python is the newline (\n). However, other characters can be used to indicate EOL. When a text editor or a program interpreter encounters the ASCII equivalent of the EOL character, it displays the remaining file contents starting from a new line. Contents in a text file are usually separated by whitespace, but comma (,) and tab (\t) are also commonly used to separate values in a text file.

Activity 2.1

Create a text file using notepad and write your name and save it. Now, create a .docx file using Microsoft Word and write your name and save it as well. Check and compare the file size of both the files. You will find that the size of .txt file is in bytes whereas that of .docx is in KBs.

2.2.2 Binary Files

Binary files are also stored in terms of bytes (0s and 1s), but unlike text files, these bytes do not represent the ASCII values of characters. Rather, they represent the actual content such as image, audio, video, compressed versions of other files, executable files, etc. These files are not human readable. Thus, trying to open a binary file using a text editor will show some garbage values. We need specific software to read or write the contents of a binary file.

Binary files are stored in a computer in a sequence of bytes. Even a single bit change can corrupt the file and make it unreadable to the supporting application. Also, it is difficult to remove any error which may occur in the binary file as the stored contents are not human readable. We can read and write both text and binary files through Python programs.

2.3 Opening and Closing a Text File

In real world applications, computer programs deal with data coming from different sources like databases, CSV files, HTML, XML, JSON, etc. We broadly access files either to write or read data from it. But operations on files include creating and opening a file, writing data in a file, traversing a file, reading data from a file and so on. Python has the io module that contains different functions for handling files.

2.3.1 Opening a file

To open a file in Python, we use the open() function. The syntax of open() is as follows:

```
file_object= open(file_name, access_mode)
```

This function returns a file object called file handle which is stored in the variable file_object. We can use this variable to transfer data to and from the file (read and write) by calling the functions defined in the Python's io module. If the file does not exist, the above statement creates a new empty file and assigns it the name we specify in the statement.

The file_object has certain attributes that tells us basic information about the file, such as:

- <file.closed> returns true if the file is closed and false otherwise.

The file_object establishes a link between the program and the data file stored in the permanent storage.

- `<file.mode>` returns the access mode in which the file was opened.

- `<file.name>` returns the name of the file.

The file_name should be the name of the file that has to be opened. If the file is not in the current working directory, then we need to specify the complete path of the file along with its name.

The access_mode is an optional argument that represents the mode in which the file has to be accessed by the program. It is also referred to as processing mode. Here mode means the operation for which the file has to be opened like <r> for reading, <w> for writing, <+> for both reading and writing, <a> for appending at the end of an existing file. The default is the read mode. In addition, we can specify whether the file will be handled as binary (<b>) or text mode. By default, files are opened in text mode that means strings can be read or written. Files containing non-textual data are opened in binary mode that means read/write are performed in terms of bytes. Table 2.1 lists various file access modes that can be used with the open() method. The file offset position in the table refers to the position of the file object when the file is opened in a particular mode.

Table 2.1   File Open Modes

File Mode	Description	File Offset position
<r>	Opens the file in read-only mode.	Beginning of the file
<rb>	Opens the file in binary and read-only mode.	Beginning of the file
<r+> or <+r>	Opens the file in both read and write mode.	Beginning of the file
<w>	Opens the file in write mode. If the file already exists, all the contents will be overwritten. If the file doesn't exist, then a new file will be created.	Beginning of the file
<wb+> or <+wb>	Opens the file in read,write and binary mode. If the file already exists, the contents will be overwritten. If the file doesn't exist, then a new file will be created.	Beginning of the file
<a>	Opens the file in append mode. If the file doesn't exist, then a new file will be created.	End of the file
<a+> or <+a>	Opens the file in append and read mode. If the file doesn't exist, then it will create a new file.	End of the file

Consider the following example.

```
myObject=open("myfile.txt", "a+")
```

In the above statement, the file *myfile.txt* is opened in append and read modes. The file object will be at the end of the file. That means we can write data at the end of the file and at the same time we can also read data from the file using the file object named *myObject.*

2.3.2 Closing a 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. The syntax of `close()` is:

```
file_object.close()
```

Here, file_object is the object that was returned while opening the file.

Python makes sure that any unwritten or unsaved data is flushed off (written) to the file before it is closed. Hence, it is always advised to close the file once our work is done. Also, if the file object is re-assigned to some other file, the previous file is automatically closed.

2.3.3 Opening a file using with clause

In Python, we can also open a file using with clause. The syntax of with clause is:

```
with open (file_name, access_mode) as file_
object:
```

The advantage of using with clause is that any file that is opened using this 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. Also, it provides a simpler syntax.

```
with open("myfile.txt","r+") as myObject:
    content = myObject.read()
```

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

2.4 WRITING TO A TEXT FILE

For writing to a file, we first need to open it in write or append mode. If we open an existing file in write mode, the previous data will be erased, and the file object will be positioned at the beginning of the file. On the other

Think and Reflect

For a newly created file, is there any difference between write() and append() methods?

hand, in append mode, new data will be added at the end of the previous data as the file object is at the end of the file. After opening the file, we can use the following methods to write data in the file.

- write() - for writing a single string

- writelines() - for writing a sequence of strings

2.4.1 The write() method

write() method takes a string as an argument and writes it to the text file. It returns the number of characters being written on single execution of the write() method. Also, we need to add a newline character (\n) at the end of every sentence to mark the end of line.

Consider the following piece of code:

```
>>> myobject=open("myfile.txt",'w')
>>> myobject.write("Hey I have started
    #using files in Python\n")
41
>>> myobject.close()
```

On execution, write() returns the number of characters written on to the file. Hence, 41, which is the length of the string passed as an argument, is displayed.

Note: '\n' is treated as a single character

If numeric data are to be written to a text file, the data need to be converted into string before writing to the file. For example:

```
>>>myobject=open("myfile.txt",'w')
>>> marks=58
#number 58 is converted to a string using
#str()
>>> myobject.write(str(marks))
2
>>>myobject.close()
```

We can also use the flush() method to clear the buffer and write contents in buffer to the file. This is how programmers can forcefully write to the file as and when required.

The `write()` actually writes data onto a buffer. When the close() method is executed, the contents from this buffer are moved to the file located on the permanent storage.

2.4.2 The writelines() method

This method is used to write multiple strings to a file. We need to pass an iterable object like lists, tuple, etc. containing strings to the `writelines()` method. Unlike

write(), the `writelines()` method does not return the number of characters written in the file. The following code explains the use of `writelines()`.

```
>>> myobject=open("myfile.txt",'w')
>>> lines = ["Hello everyone\n", "Writing
    #multiline strings\n", "This is the
    #third line"]
>>> myobject.writelines(lines)
>>>myobject.close()
```

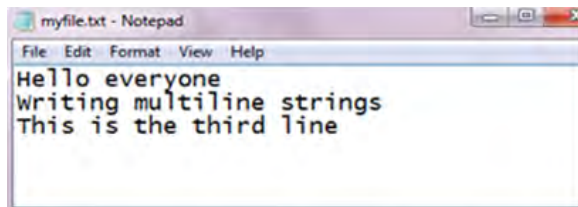On opening myfile.txt, using notepad, its content will appear as shown in Figure 2.1.

Figure 2.1: Contents of myfile.txt

2.5 READING FROM A TEXT FILE

We can write a program to read the contents of a 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 contents of a file:

2.5.1 The read() method

This method is used to read a specified number of bytes of data from a data file. The syntax of read() method is:

```
file_object.read(n)
```

Consider the following set of statements to understand the usage of read() method:

```
>>>myobject=open("myfile.txt",'r')
>>> myobject.read(10)
'Hello ever'
>>> myobject.close()
```

If no argument or a negative number is specified in read(), the entire file content is read. For example,

```
>>> myobject=open("myfile.txt",'r')
>>> print(myobject.read())
Hello everyone
Writing multiline strings
This is the third line
>>> myobject.close()
```

2.5.2 The readline([n]) method

This method reads one complete line from a file where each line terminates with a newline (\n) character. It can also be used to read a specified number (n) of bytes of data from a file but maximum up to the newline character (\n). In the following example, the second statement reads the first ten characters of the first line of the text file and displays them on the screen.

```
>>> myobject=open("myfile.txt",'r')
>>> myobject.readline(10)
'Hello ever'
>>> myobject.close()
```

If no argument or a negative number is specified, it reads a complete line and returns string.

```
>>>myobject=open("myfile.txt",'r')
>>> print (myobject.readline())
'Hello everyone\n'
```

To read the entire file line by line using the readline(), we can use a loop. This process is known as looping/iterating over a file object. It returns an empty string when EOF is reached.

2.5.3 The readlines() method

The method reads all the lines and returns the lines along with newline as a list of strings. The following example uses readlines() to read data from the text file *myfile.txt*.

```
>>> myobject=open("myfile.txt", 'r')
>>> print(myobject.readlines())
['Hello everyone\n', 'Writing multiline
strings\n', 'This is the third line']
>>> myobject.close()
```

As shown in the above output, when we read a file using readlines() function, lines in the file become members of a list, where each list element ends with a newline character ('\n').

In case we want to display each word of a line separately as an element of a list, then we can use split() function. The following code demonstrates the use of split() function.

```
>>> myobject=open("myfile.txt",'r')
>>> d=myobject.readlines()
```

```
>>> for line in d:
        words=line.split()
        print(words)


['Hello', 'everyone']
['Writing', 'multiline', 'strings']
['This', 'is', 'the', 'third', 'line']
```

In the output, each string is returned as elements of a list. However, if *splitlines()* is used instead of split(), then each line is returned as element of a list, as shown in the output below:

```
>>> for line in d:
        words=line.splitlines()
        print(words)


['Hello everyone']
['Writing multiline strings']
['This is the third line']
```

Let us now write a program that accepts a string from the user and writes it to a text file. Thereafter, the same program reads the text file and displays it on the screen.

Program 2-1   Writing and reading to a text file

```
fobject=open("testfile.txt","w")      # creating a data file
sentence=input("Enter the contents to be written in the file: ")
fobject.write(sentence)               # Writing data to the file
fobject.close()                       # Closing a file

print("Now reading the contents of the file: ")
fobject=open("testfile.txt","r")
#looping over the file object to read the file
for str in fobject:
        print(str)
fobject.close()
```

In Program 2.1, the file named *testfile.txt* is opened in write mode and the file handle named *fobject* is returned. The string is accepted from the user and written in the file using write(). Then the file is closed and again opened in read mode. Data is read from the file and displayed till the end of file is reached.

Output of Program 2-1:

```
>>>
 RESTART: Path_to_file\Program2-1.py
Enter the contents to be written in the file:
roll_numbers = [1, 2, 3, 4, 5, 6]
Now reading the contents of the file:
roll_numbers = [1, 2, 3, 4, 5, 6]
>>>
```

2.6 SETTING OFFSETS IN A FILE

The functions that we have learnt till now are used to access the data sequentially from a file. But if we want to access data in a random fashion, then Python gives us seek() and tell() functions to do so.

2.6.1 The tell() method

This function returns an integer that specifies the current position of the file object in the file. The position so specified is the byte position from the beginning of the file till the current position of the file object. The syntax of using tell() is:

```
file_object.tell()
```

2.6.2 The seek() method

This method is used to position the file object at a particular position in a file. The syntax of seek() is:

```
file_object.seek(offset [, reference_point])
```

In the above syntax, offset is the number of bytes by which the file object is to be moved. reference_point indicates the starting position of the file object. That is, with reference to which position, the offset has to be counted. It can have any of the following values:

0 - beginning of the file
1 - current position of the file
2 - end of file

By default, the value of reference_point is 0, i.e. the offset is counted from the beginning of the file. For example, the statement fileObject.seek(5,0) will position the file object at 5th byte position from the beginning of the file. The code in Program 2-2 below demonstrates the usage of seek() and tell().

Think and Reflect

Does the seek() function work in the same manner for text and binary files?

Program 2-2   Application of seek() and tell()

```
print("Learning to move the file object")
fileobject=open("testfile.txt","r+")
str=fileobject.read()
print(str)
print("Initially, the position of the file object is: ",fileobject.
tell())
fileobject.seek(0)
print("Now the file object is at the beginning of the file:
",fileobject.tell())
fileobject.seek(10)
print("We are moving to 10th byte position from the beginning of
file")
print("The position of the file object is at", fileobject.tell())
str=fileobject.read()
print(str)
```

Output of Program 2-2:

```
>>>
 RESTART: Path_to_file\Program2-2.py
Learning to move the file object
roll_numbers = [1, 2, 3, 4, 5, 6]
Initially, the position of the file object is: 33
Now the file object is at the beginning of the file: 0
We are moving to 10th byte position from the beginning of file

The position of the file object is at 10
rs = [1, 2, 3, 4, 5, 6]
>>>
```

2.7 CREATING AND TRAVERSING A TEXT FILE

Having learnt various methods that help us to open and close a file, read and write data in a text file, find the position of the file object and move the file object at a desired location, let us now perform some basic operations on a text file. To perform these operations, let us assume that we will be working with practice.txt.

2.7.1 Creating a file and writing data

To create a text file, we use the open() method and provide the filename and the mode. If the file already exists with the same name, the open() function will behave differently depending on the mode (write or append) used. If it is in write mode (w), then all the existing contents of file will be lost, and an empty file will be created with the same name. But, if the file is

created in append mode (a), then the new data will be written after the existing data. In both cases, if the file does not exist, then a new empty file will be created. In Program 2-3, a file, *practice.txt* is opened in write (w) mode and three sentences are stored in it as shown in the output screen that follows it

Program 2-3 To create a text file and write data in it

```
# program to create a text file and add data
fileobject=open("practice.txt","w+")
while True:
    data= input("Enter data to save in the text file: ")
    fileobject.write(data)
    ans=input("Do you wish to enter more data?(y/n): ")
    if ans=='n': break
fileobject.close()
```

Output of Program 2-3:
```
>>>
 RESTART: Path_to_file\Program2-3.py
Enter data to save in the text file: I am interested to learn about
Computer Science
Do you wish to enter more data?(y/n): y
Enter data to save in the text file: Python is easy to learn
Do you wish to enter more data?(y/n): n
>>>
```

2.7.2 Traversing a file and displaying data

To read and display data that is stored in a text file, we will refer to the previous example where we have created the file *practice.txt*. The file will be opened in read mode and reading will begin from the beginning of the file.

Program 2-4 To display data from a text file

```
fileobject=open("practice.txt","r")
str = fileobject.readline()
while str:
    print(str)
    str=fileobject.readline()
fileobject.close()
```

In Program 2-4, the readline() is used in the while loop to read the data line by line from the text file. The lines are displayed using the print(). As the end of file is reached, the readline() will return an empty string. Finally, the file is closed using the close().

Output of Program 2-4:

```
>>>
I am interested to learn about Computer SciencePython is easy to learn
```

Till now, we have been creating separate programs for writing data to a file and for reading the file. Now let us create one single program to read and write data using a single file object. Since both the operations have to be performed using a single file object, the file will be opened in w+ mode.

Program 2-5   To perform reading and writing operation in a text file

```
fileobject=open("report.txt", "w+")
print ("WRITING DATA IN THE FILE")
print() # to display a blank line
while True:
    line= input("Enter a sentence ")
    fileobject.write(line)
    fileobject.write('\n')
    choice=input("Do you wish to enter more data? (y/n): ")
    if choice in ('n','N'): break
print("The byte position of file object is ",fileobject.tell())
fileobject.seek(0)  #places file object at beginning of file
print()
print("READING DATA FROM THE FILE")
str=fileobject.read()
print(str)
fileobject.close()
```

In Program 2-5, the file will be read till the time end of file is not reached and the output as shown in below is displayed.

Output of Program 2-5:

```
>>>
 RESTART: Path_to_file\Program2-5.py
WRITING DATA IN THE FILE

Enter a sentence I am a student of class XII
Do you wish to enter more data? (y/n): y
Enter a sentence my school contact number is 4390xxx8
Do you wish to enter more data? (y/n): n
The byte position of file object is  67

READING DATA FROM THE FILE
I am a student of class XII
my school contact number is 4390xxx8
>>>
```

2.8 THE PICKLE MODULE

We know that Python considers everything as an object. So, all data types including list, tuple, dictionary, etc. are also considered as objects. During execution of a program, we may require to store current state of variables so that we can retrieve them later to its present state. Suppose you are playing a video game, and after some time, you want to close it. So, the program should be able to store the current state of the game, including current level/stage, your score, etc. as a Python object. Likewise, you may like to store a Python dictionary as an object, to be able to retrieve later. To save any object structure along with data, Python provides a module called Pickle. The module Pickle is used for serializing and de-serializing any Python object structure. Pickling is a method of preserving food items by placing them in some solution, which increases the shelf life. In other words, it is a method to store food items for later consumption.

Serialization is the process of transforming data or an object in memory (RAM) to a stream of bytes called byte streams. These byte streams in a binary file can then be stored in a disk or in a database or sent through a network. Serialization process is also called pickling.

De-serialization or unpickling is the inverse of pickling process where a byte stream is converted back to Python object.

The pickle module deals with binary files. Here, data are not written but dumped and similarly, data are not read but loaded. The Pickle Module must be imported to load and dump data. The pickle module provides two methods - dump() and load() to work with binary files for pickling and unpickling, respectively.

2.8.1 The dump() method

This method is used to convert (pickling) Python objects for writing data in a binary file. The file in which data are to be dumped, needs to be opened in binary write mode (wb).

Syntax of dump() is as follows:

```
dump(data_object, file_object)
```

where data_object is the object that has to be dumped to the file with the file handle named file_

object. For example, Program 2-6 writes the record of a student (roll_no, name, gender and marks) in the binary file named mybinary.dat using the dump(). We need to close the file after pickling.

Program 2-6   Pickling data in Python

```
import pickle
listvalues=[1,"Geetika",'F', 26]
fileobject=open("mybinary.dat", "wb")
pickle.dump(listvalues,fileobject)
fileobject.close()
```

2.8.2 The load() method

This method is used to load (unpickling) data from a binary file. The file to be loaded is opened in binary read (rb) mode. Syntax of load() is as follows:

Store_object = load(file_object)

Here, the pickled Python object is loaded from the file having a file handle named file_object and is stored in a new file handle called store_object. The program 2-7 demonstrates how to read data from the file *mybinary.dat* using the load().

Program 2-7   Unpickling data in Python

```
import pickle
print("The data that were stored in file are: ")
fileobject=open("mybinary.dat","rb")
objectvar=pickle.load(fileobject)
fileobject.close()
print(objectvar)
```

Output of Program 2-7:

```
>>>
 RESTART: Path_to_file\Program2-7.py
The data that were stored in file are:
[1, 'Geetika', 'F', 26]
>>>
```

2.8.3 File handling using pickle module

As we read and write data in a text file, similarly we will be adding and displaying data for a binary file. Program 2-8 accepts a record of an employee from the user and appends it in the binary file *tv*. Thereafter, the records are read from the binary file and displayed on the screen using the same object. The user may enter

as many records as they wish to. The program also displays the size of binary files before starting with the reading process.

Program 2-8 To perform basic operations on a binary file using pickle module

```
# Program to write and read employee records in a binary file
import pickle
print("WORKING WITH BINARY FILES")
bfile=open("empfile.dat","ab")
recno=1
print ("Enter Records of Employees")
print()
#taking data from user and dumping in the file as list object
while True:
    print("RECORD No.", recno)
    eno=int(input("\tEmployee number : "))
    ename=input("\tEmployee Name : ")
    ebasic=int(input("\tBasic Salary : "))
    allow=int(input("\tAllowances : "))
    totsal=ebasic+allow
    print("\tTOTAL SALARY : ", totsal)
    edata=[eno,ename,ebasic,allow,totsal]
    pickle.dump(edata,bfile)
    ans=input("Do you wish to enter more records (y/n)? ")
    recno=recno+1
    if ans.lower()=='n':
        print("Record entry OVER ")
        print()
        break
# retrieving the size of file
print("Size of binary file (in bytes):",bfile.tell())
bfile.close()
# Reading the employee records from the file using load() module
print("Now reading the employee records from the file")
print()
readrec=1
try:
  with open("empfile.dat","rb") as bfile:
  while True:
        edata=pickle.load(bfile)
        print("Record Number : ",readrec)
        print(edata)
        readrec=readrec+1
except EOFError:
    pass
bfile.close()
```

Output of Program 2-8:

```
>>>
 RESTART: Path_to_file\Program2-8.py
WORKING WITH BINARY FILES
Enter Records of Employees

RECORD No. 1
        Employee number : 11
        Employee Name : D N Ravi
        Basic Salary : 32600
        Allowances : 4400
        TOTAL SALARY :  37000
Do you wish to enter more records (y/n)? y
RECORD No. 2
        Employee number : 12
        Employee Name : Farida Ahmed
        Basic Salary : 38250
        Allowances : 5300
        TOTAL SALARY :  43550
Do you wish to enter more records (y/n)? n
Record entry OVER

Size of binary file (in bytes): 216
Now reading the employee records from the file

Record Number :  1
[11, 'D N Ravi', 32600, 4400, 37000]
Record Number :  2
[12, 'Farida Ahmed', 38250, 5300, 43550]
>>>
```

As each employee record is stored as a list in the file empfile.dat, hence while reading the file, a list is displayed showing record of each employee. Notice that in Program 2-8, we have also used try.. except block to handle the end-of-file exception.

SUMMARY

- A file is a named location on a secondary storage media where data are permanently stored for later access.

- A text file contains only textual information consisting of alphabets, numbers and other

special symbols. Such files are stored with extensions like *.txt, .py, .c, .csv, .html,* etc. Each byte of a text file represents a character.

- Each line of a text file is stored as a sequence of ASCII equivalent of the characters and is terminated by a special character, called the End of Line (EOL).

- Binary file consists of data stored as a stream of bytes.

- open() method is used to open a file in Python and it returns a file object called file handle. The file handle is used to transfer data to and from the file by calling the functions defined in the Python's io module.

- close() method is used to close the file. While closing a file, the system frees up all the resources like processor and memory allocated to it.

- write() method takes a string as an argument and writes it to the text file.

- writelines() method is used to write multiple strings to a file. We need to pass an iterable object like lists, tuple etc. containing strings to writelines() method.

- read([n]) method is used to read a specified number of bytes (n) of data from a data file.

- readline([n]) method reads one complete line from a file where lines are ending with a newline (\n). It can also be used to read a specified number (n) of bytes of data from a file but maximum up to the newline character (\n).

- readlines() method reads all the lines and returns the lines along with newline character, as a list of strings.

- tell() method returns an integer that specifies the current position of the file object. The position so specified is the byte position from the beginning of the file till the current position of the file object.

- seek() method is used to position the file object at a particular position in a file.

- Pickling is the process by which a Python object is converted to a byte stream.
- `dump()` method is used to write the objects in a binary file.
- `load()` method is used to read data from a binary file.

❓ EXERCISE

1. Differentiate between:
   a) text file and binary file
   b) readline() and readlines()
   c) write() and writelines()

2. Write the use and syntax for the following methods:
   a) open()
   b) read()
   c) seek()
   d) dump()

3. Write the file mode that will be used for opening the following files. Also, write the Python statements to open the following files:
   a) a text file "example.txt" in both read and write mode
   b) a binary file "bfile.dat" in write mode
   c) a text file "try.txt" in append and read mode
   d) a binary file "btry.dat" in read only mode.

4. Why is it advised to close a file after we are done with the read and write operations? What will happen if we do not close it? Will some error message be flashed?

5. What is the difference between the following set of statements (a) and (b):
   a) P = open("practice.txt","r")
       P.read(10)
   b) with open("practice.txt", "r") as P:
           x = P.read()

6. Write a command(s) to write the following lines to the text file named *hello.txt*. Assume that the file is opened in append mode.
   " Welcome my class"
   "It is a fun place"
   "You will learn and play"

7. Write a Python program to open the file *hello.txt* used in question no 6 in read mode to display its contents. What will be the difference if the file was opened in write mode instead of append mode?

8. Write a program to accept string/sentences from the user till the user enters "END" to. Save the data in a text file and then display only those sentences which begin with an uppercase alphabet.

9. Define pickling in Python. Explain serialization and deserialization of Python object.

10. Write a program to enter the following records in a binary file:

    Item No   integer
    Item_Name   string
    Qty   integer
    Price   float

    Number of records to be entered should be accepted from the user. Read the file to display the records in the following format:

    Item No:
    Item Name :
    Quantity:
    Price per item:
    Amount: (to be calculated as Price * Qty)