K. J. Somaiya College of Engineering, Mumbai-77

(A Constituent College of Somaiya Vidyavihar University)

Batch:	**B2**	**Roll No.:**	**16010121110**

Experiment / assignment / tutorial No. 10

Grade: AA / AB / BB / BC / CC / CD /DD

TITLE: File Handling

AIM: Write a program to create a file rollcall.txt which stores the student details by adding their Roll_No,Name and Department into it.

Expected OUTCOME of Experiment:

CO1: Use basic data structures in Python
CO2: Use different Decision Making statements and Functions in Python.
CO4: Implement different File handling operations

Resource Needed: Python IDE

Theory:

Python provides the basic functions and methods necessary to manipulate files by default. You can do most file manipulation using a file object. We cannot express the data to be stored in a variable all the time because the variables are volatile in nature. So, to handle such situations, the role of files comes into the picture. Since files are non-volatile in nature, the data will be stored permanently in a secondary device (e.g., Hard Disk) and handled with Python in our applications. We do the operations on files in Python using some built-in methods or functions.

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.

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.
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.

With the Python programming language, we can handle both text files and binary files.

The following are the operations on files:

1. **Open a file**

Python has an in-built function called open() that opens a file:

open('filename', mode)

• filename: Gives the name of the file that the file object has opened.
• mode: An attribute of a file object that tells you the mode in which a file was opened.
"r" - Read - Default value. Opens a file for reading, error if the file does not exist
"a" - Append - Opens a file for appending, creates the file if it does not exist
"w" - Write - Opens a file for writing, creates the file if it does not exist
"x" - Create - Creates the specified file, returns an error if the file exists

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.

3. **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 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():

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.

• writeline()

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 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()
```

4. 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:

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()
```

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()
```

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()
```

5. 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.

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()

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.

Example:

```
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(5)
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:

```
>>>
START: 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
numbers = [1, 2, 3, 4, 5, 6]
```

Problem Definition:

---

1. Write a program to create a file rollcall.txt which stores the student details by adding their Roll_No,Name and Department into it using following format:

Roll No	Name	Department
1601001	Abc	Computer
1601003	Xyz	IT

Obtain the details for rollcall list from the user.

Books/ Journals/ Websites referred:

1. **Reema Thareja , "Python Programming: Using Problem Solving Approach", Oxford University Press, First Edition 2017, India**
2. **Sheetal Taneja and Naveen Kumar," Python Programing: A Modular Approach", Pearson India, Second Edition 2018, India**

Implementation details:

```
[open("rollcall.txt","a").write("\t\t\t".join([input("Please enter Roll number "),input("Please enter name "),input("Please enter department "),"\n"])) for i in range(0,3)] # one line solution
```

Output(s):

main.py rollcall.txt

```
1  Roll no.     Name          Department
2  110          Aatmaj            Comps
3  111          Aditya            Comps
4  123          Sushant           IT
5
```

```
Please enter Roll number 110
Please enter name Aatmaj
Please enter department Comps
Please enter Roll number 111
Please enter name Aditya
Please enter department Comps
Please enter Roll number 123
Please enter name Sushant
Please enter department IT
```

Conclusion:

Thus we have understood how file handling works in python. We have understood the four modes of opening file in python and their respective uses. File handling is very useful for long term storage of data. Most common types of data are csv files and json files which have larger implications and usage across all technologies.

Post Lab Descriptive Questions:

1. Write a program that prompt the user for a file name and then read and prints the contents of the requested file in the upper case.

```
try:
   file=open(input("Please enter a valid file name "),"r")
   for i in file:
      print(i.upper())
except FileNotFoundError:
   print("please enter  valid file name ")
```

2. 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?

No. But sometimes errors may be caused due to it. Rather than suspecting lack of close statement for errors it is advisable to close the file whenever appropriate.

3. Which of the following command is used to open a file "c:\temp.txt" in read-mode only?

A. infile = open("c:\temp.txt", "r")
B. infile = open("c:\\temp.txt", "r")
C. infile = open(file = "c:\temp.txt", "r+")
D. infile = open(file = "c:\\temp.txt", "r+")

B

Date: 15 July 2022 **Signature of faculty in-charge**