Course Name: Fundamentals of Programming & Data Science	**Course Code:** CMPE-112L
Assignment Type: Lab	**Dated:** 25th March 2024
Semester: 2nd	**Session:** 2023
Lab/Project/Assignment #: 8	**CLOs to be covered: CLO 1**
Lab Title: File Handling	**Teacher Name:** Engr. Afeef Obaid

Lab Evaluation:

CLO 1	Apply appropriate programming techniques to create executable programs to solve well defined problems					
Levels (Marks)	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Level 6**
(10)						
					Total	**/10**

Rubrics for Current Lab Evaluation

Scale	**Marks**	**Level**	**Rubric**
Excellent	**10**	L1	Submitted all lab tasks, BONUS task, have good understanding.
Very Good	**8**	L2	Submitted the lab tasks but have good understanding
Good	**6**	L3	Submitted the lab tasks but have weak understanding.
Basic	**4**	L4	Submitted the lab tasks but have no understanding.
Barely Acceptable	**2**	L5	Submitted only one lab task.
Not Acceptable	**0**	L6	Did not attempt

LAB No. 8

Lab Goals/Objectives:

By reading this manual, students will be able:

- To understand the concept of File handling in Python
- To Become familiar with the concept of read, write and append mode in Python
- To learn about `with` Keyword in Python
- To learn about OS modules in Python

Equipment Required: Computer system with Pycharm IDE and python package installed on it

File Handling

The file handling plays an important role when the data needs to be stored permanently into the file. A file is a named location on disk to store related information. We can access the stored information (non-volatile) after the program termination.

The file-handling implementation is slightly lengthy or complicated in the other programming language, but it is easier and shorter in Python.

In Python, files are treated in two modes as text or binary. The file may be in the text or binary format, and each line of a file is ended with the special character.

Hence, a file operation can be done in the following order.

- Open a file
- Read or write - Performing operation
- Close the file

Open a file

Before performing any operation on the file like reading or writing, first, we have to open that file. For this, we should use Python's inbuilt function open() but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

Syntax:

f = open(filename, mode)

Sr.	**Mode**	**Description**
1	r	It opens the file to read-only mode. The file pointer exists at the beginning. The file is by default open in this mode if no access mode is passed.
2	rb	It opens the file to read-only in binary format. The file pointer exists at the beginning of the file.

3	r+	It opens the file to read and write both. The file pointer exists at the beginning of the file.
4	rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
5	w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name. The file pointer exists at the beginning of the file.
6	wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists. The file pointer exists at the beginning of the file.
7	w+	It opens the file to write and read both. It is different from r+ in the sense that it overwrites the previous file if one exists whereas r+ doesn't overwrite the previously written file. It creates a new file if no file exists. The file pointer exists at the beginning of the file.
8	wb+	It opens the file to write and read both in binary format. The file pointer exists at the beginning of the file.
9	a	It opens the file in the append mode. The file pointer exists at the end of the previously written file if exists any. It creates a new file if no file exists with the same name.
10	ab	It opens the file in the append mode in binary format. The pointer exists at the end of the previously written file. It creates a new file in binary format if no file exists with the same name.
11	a+	It opens a file to append and read both. The file pointer remains at the end of the file if a file exists. It creates a new file if no file exists with the same name.
12	ab+	It opens a file to append and read both in binary format. The file pointer remains at the end of the file.

Example

```
fileptr = open("file.txt", "r")
if fileptr:
    print("file is opened successfully")
```

In the above example you will get the error as “file.txt” not exist.

For creating “file.txt” run the above code in the write (w) mode

```
fileptr = open("file.txt", "w")
if fileptr:
   print("file is opened successfully")
```

In the above code, we have passed filename as a first argument and opened file in write mode as we mentioned **w** as the second argument. The fileptr holds the file object and if the file is opened successfully, it will execute the print statement

Writing the file

To write some text to a file, we need to open the file using the open method with one of the following access modes.

w: It will overwrite the file if any file exists. The file pointer is at the beginning of the file.

a: It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.

Example # 1

```
fileptr = open("file.txt", "w")
if fileptr:
   print("file is opened successfully")
fileptr.write('''Python is the modern day language.
It makes things so simple.
It is the fastest-growing programing language.''')
fileptr.close()
```

Example # 2

```
fileptr = open("file.txt", "a")
if fileptr:
   print("file is opened successfully")
fileptr.write("\nPython has an easy syntax and user-friendly interaction.")
fileptr.close()
```

We can see that the content of the file is modified. We have opened the file in a mode and it appended the content in the existing **"file.txt"**.

Read the file

To read a file using the Python script, the Python provides the read() method. The read() method reads a string from the file. It can read the data in the text as well as a binary format.

Syntax:

```
fileobj.read(<count>)
```

Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

Example

```
fileptr = open("file.txt", "r")
if fileptr:
   print("file is opened successfully")
content=fileptr.read(15)
print(type(content))
print(content)
completecontent=fileptr.read()
print(completecontent)
fileptr.close()
```

Read the file through loop

We can read the file using for loop.

Example

```
fileptr = open("file.txt", "r")
if fileptr:
   print("file is opened successfully")
for line in fileptr:
   print(line,end="")
```

Read Line of the file

Python facilitates to read the file line by line by using a function readline() method. The readline() method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

Example

```
fileptr = open("file.txt", "r")
if fileptr:
   print("file is opened successfully")
print(fileptr.readline())
print(fileptr.readline())
fileptr.close()
```

Read Lines of the file

Python provides also the readlines() method which is used for the reading lines. It returns the list of the lines till the end of file(EOF) is reached.

Example

```
fileptr = open("file.txt", "r")
if fileptr:
   print("file is opened successfully")
```

```
lines=fileptr.readlines()
print(lines)
for line in lines:
   print(line,end="")
fileptr.close()
```

Closing the file

Once all the operations are done on the file, we must close it through our Python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.

We can perform any operation on the file externally using the file system which is the currently opened in Python; hence it is good practice to close the file once all the operations are done.

Syntax:

```
fileobject.close()
```

Example

```
fileptr = open("file.txt", "r")
if fileptr:
   print("file is opened successfully")
fileptr.close()
```

After closing the file, we cannot perform any operation in the file. The file needs to be properly closed. If any exception occurs while performing some operations in the file then the program terminates without closing the file.
We should use the following method to overcome such type of problem.

```
try:
  fileptr = open("file.txt")
  if fileptr:
     print("File open successfully")
finally:
  fileptr.close()
```

The with statement

The with statement was introduced in python 2.5. The with statement is useful in the case of manipulating the files. It is used in the scenario where a pair of statements is to be executed with a block of code in between.

Syntax:

```
with open(<file name>, <access mode>) as <file-pointer>:
```

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the with statement in the case of files because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file, we don't need to write the close() function. It doesn't let the file to corrupt.

Example

```
with open("file.txt",'r') as fileptr:
    if fileptr:
        print("File open successfully")
        content = fileptr.read();
        print(content)
```

File Pointer positions

Python provides the tell() method which is used to print the byte number at which the file pointer currently exists.

Example

```
with open("file.txt",'r') as fileptr:
    if fileptr:
        print("File open successfully")
        print("The filepointer is at byte :", fileptr.tell())
        content = fileptr.read();
        print(content)
        print("After reading, the filepointer is at:", fileptr.tell())
```

Modifying file pointer position

In real-world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.

For this purpose, the Python provides us the seek() method which enables us to modify the file pointer position externally.

Syntax:

<file-ptr>.seek(offset[, from)

The seek() method accepts two parameters:

offset: It refers to the new position of the file pointer within the file.

from: It indicates the reference position from where the bytes are to be moved. If it is set to 0, the beginning of the file is used as the reference position. If it is set to 1, the current position of the file pointer is used as the reference position. If it is set to 2, the end of the file pointer is used as the reference position.

Example

```
with open("file.txt",'r') as fileptr:
    if fileptr:
        print("File open successfully")
        print("The filepointer is at byte :", fileptr.tell())
        fileptr.seek(10)
        content = fileptr.read();
        print(content)
        print("After reading, the filepointer is at:", fileptr.tell())
```

Python OS module

The os module provides the functions that are involved in file processing operations like renaming, deleting, etc.

Renaming the file

The os module provides us the rename() method to rename the specified file to a new name.

Syntax:

```
os.rename(current-name, new-name)
```

The first argument is the current file name and the second argument is the modified name.

Example

```
import os
os.rename("file.txt", "file1.txt")
```

Removing the file

The os module provides the remove() method which is used to remove the specified file.

Syntax:

```
os. remove(file-name)
```

Example

```
import os;
fileptr=open("test.txt","x")
fileptr.close()
os.remove("test.txt")
```

Lab Tasks

- Write the python program that takes the 5 lines from the user and store it into the file named as `input.txt` and then read each line from the file and reverse its each word and store in into another file named as `output.txt` with line number.

- Write the python program that keep the record of attendance of all the registered students in the Lab of Fundamentals of Programming and Data Science. It also have the feature to tell whether the particular student is eligible to sit in exam or not.

- Write a Python program that encrypts the contents of a text file, saves the encrypted text to another file names as `decrypt.txt` and a program to decrypt the text in `decrypt.txt`.

- Write a Python program that creates a backup of a given file by copying its contents to another file with a timestamp appended to its name.

- Write a Python program that searches for a specific keyword or pattern in a directory of text files and displays the file names and line numbers where the keyword or pattern is found.