

LAB Manual

PART A

(PART A : TO BE REFERRED BY STUDENTS)

Experiment No.04

A.1 Aim:

To explore Files and directories

- a. Python program to append data to existing file and then display the entire file
- b. Python program to count number of lines, words and characters in a file.
- c. Python program to display file available in current directory

A.2 Prerequisite:

1. C, JAVA Language

A.3 Outcome:

After successful completion of this experiment students will be able to

- . To explore contents of files, directories and text processing with python .

A.4 Theory& Procedure:

Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.

The open Function

Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details –

- file_name – The file_name argument is a string value that contains the name of the file that you want to access.

- `access_mode` – The `access_mode` determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- `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. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file –

Sr.No.	Modes & Description
1	<p>r</p> <p>Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.</p>
2	<p>rb</p> <p>Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.</p>
3	<p>r+</p> <p>Opens a file for both reading and writing. The file pointer placed at the beginning of the file.</p>
4	<p>rb+</p> <p>Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.</p>
5	<p>w</p> <p>Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.</p>
6	<p>wb</p> <p>Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.</p>
7	<p>w+</p>

	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8	wb+ Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9	a Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10	ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
11	a+ Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12	ab+ Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The file Object Attributes

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object –

Sr.No.	Attribute & Description
1	file.closed Returns true if file is closed, false otherwise.

2	file.mode Returns access mode with which file was opened.
3	file.name Returns name of the file.
4	file.softspace Returns false if space explicitly required with print, true otherwise.

Example

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

This produces the following result –

```
Name of the file: foo.txt
Closed or not : False
Opening mode : wb
Softspace flag : 0
```

The close() Method

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

```
fileObject.close();
```

Example

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name

# Close opened file
fo.close()
```

This produces the following result –

```
Name of the file: foo.txt
```

Reading and Writing Files

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

The write() Method

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string –

Syntax

```
fileObject.write(string);
```

Here, passed parameter is the content to be written into the opened file.

Example

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");
```

```
# Close opened file
```

```
fo.close()
```

The above method would create foo.txt file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

```
Python is a great language.  
Yeah its great!!
```

The read() Method

The read() method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

Example

Let's take a file foo.txt, which we created above.

```
#!/usr/bin/python
```

```
# Open a file
```

```
fo = open("foo.txt", "r+")
```

```
str = fo.read(10);
```

```
print "Read String is : ", str
```

```
# Close opened file
```

```
fo.close()
```

This produces the following result –

```
Read String is : Python is
```

File Positions

The tell() method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The seek(offset[, from]) method changes the current file position. The offset argument indicates the number of bytes to be moved. The from argument specifies the reference position from where the bytes are to be moved.

If from is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example

Let us take a file foo.txt, which we created above.

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "r+")
str = fo.read(10);
print "Read String is : ", str

# Check current position
position = fo.tell();
print "Current file position : ", position

# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str

# Close opened file
fo.close()
```

This produces the following result –

```
Read String is : Python is
Current file position : 10
Again read String is : Python is
```

Renaming and Deleting Files

Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

The rename() Method

The rename() method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file test1.txt –

```
#!/usr/bin/python

import os

# Rename a file from test1.txt to test2.txt

os.rename( "test1.txt", "test2.txt" )
```

The remove() Method

You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

Example

Following is the example to delete an existing file test2.txt –

```
#!/usr/bin/python

import os

# Delete file test2.txt

os.remove("text2.txt")
```


Directories in Python

All files are contained within various directories, and Python has no problem handling these too. The `os` module has several methods that help you create, remove, and change directories.

The `mkdir()` Method

You can use the `mkdir()` method of the `os` module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

```
os.mkdir("newdir")
```

Example

Following is the example to create a directory `test` in the current directory –

```
#!/usr/bin/python

import os

# Create a directory "test"

os.mkdir("test")
```

The `chdir()` Method

You can use the `chdir()` method to change the current directory. The `chdir()` method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

```
os.chdir("newdir")
```

Example

Following is the example to go into `"/home/newdir"` directory –

```
#!/usr/bin/python

import os

# Changing a directory to "/home/newdir"

os.chdir("/home/newdir")
```

The `getcwd()` Method

The `getcwd()` method displays the current working directory.

Syntax

```
os.getcwd()
```

Example

Following is the example to give current directory –

```
#!/usr/bin/python

import os

# This would give location of the current directory

os.getcwd()
```

The rmdir() Method

The rmdir() method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax

```
os.rmdir('dirname')
```

Example

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

```
#!/usr/bin/python

import os

# This would remove "/tmp/test" directory.

os.rmdir( "/tmp/test" )
```

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows –

- File Object Methods: The file object provides functions to manipulate files.
- OS Object Methods: This provides methods to process files as well as directories.

PART B

Roll No. B81	Name: Bhushan Prashant Ghevde.
Class : SE - B	Batch : B4
Date of Experiment: 22/04/2021	Date of Submission: 22/04/2021
Grade :	

B.1 Document created by the student:

A. Python program to append data to existing file and then display the entire file.

Code :-

```
firstfile = input("Enter the name of first file ")
```

```
secondfile = input("Enter the name of second file ")
```

```
# opening both files in read only mode to read initial contents
```

```
f1 = open(firstfile, 'r')
```

```
f2 = open(secondfile, 'r')
```

```
# printing the contents of the file before appending
```

```
print('content of first file before appending -', f1.read())
```

```
print('content of second file before appending -', f2.read())
```

```
# closing the files
```

```
f1.close()
```

```
f2.close()
```

```
# opening first file in append mode and second file in read mode
```

```
f1 = open(firstfile, 'a+')
```

```
f2 = open(secondfile, 'r')
```

appending the contents of the second file to the first file

```
f1.write(f2.read())
```

relocating the cursor of the files at the beginning

```
f1.seek(0)
```

```
f2.seek(0)
```

printing the contents of the files after appending

```
print('content of first file after appending -', f1.read())
```

```
print('content of second file after appending -', f2.read())
```

closing the files

```
f1.close()
```

```
f2.close()
```

Output :-

Case 1:

Output:

Enter file to be read from: test.txt

Enter file to be appended to: test1.txt

Contents of file test.txt:

Appending!!

Contents of file test1.txt (before appending):

Original

Contents of file test1.txt (after appending):

Original Appending!!

Case 2:

Enter file to be read from: out.txt

Enter file to be appended to: out1.txt

Contents of file test.txt:

world

Contents of file test1.txt (before appending):

Hello

Contents of file test1.txt (after appending):

Hello world

B. Python program to count number of lines, words and characters in a file.

Code :-

```
file = open("sample.txt", "r")

number_of_lines = 0

number_of_words = 0

number_of_characters = 0

for line in file:

    line = line.strip("\n")

    words = line.split()

    number_of_lines += 1

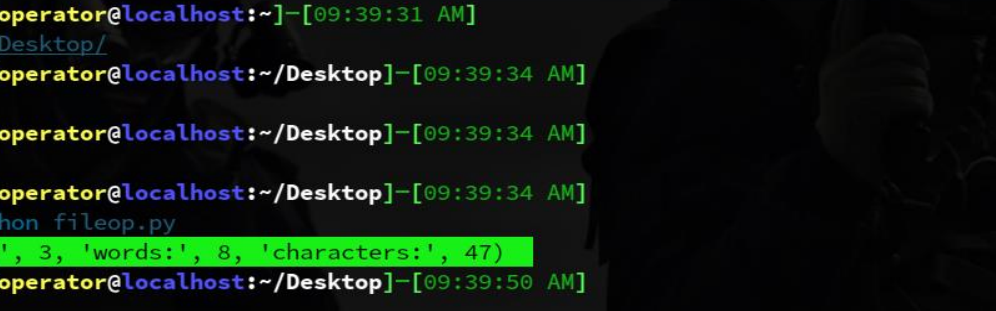
    number_of_words += len(words)

    number_of_characters += len(line)

file.close()

print("lines:", number_of_lines, "words:", number_of_words, "characters:", number_of_characters)
```

Output :-



```
Desktop : fish — Konsole
File Edit View Bookmarks Settings Help

[tier1operator@localhost:~]-[09:39:31 AM]
>$ cd Desktop/
[tier1operator@localhost:~/Desktop]-[09:39:34 AM]
>$
[tier1operator@localhost:~/Desktop]-[09:39:34 AM]
>$
[tier1operator@localhost:~/Desktop]-[09:39:34 AM]
>$ python fileop.py
('lines:', 3, 'words:', 8, 'characters:', 47)
[tier1operator@localhost:~/Desktop]-[09:39:50 AM]
>$
```

C. Python program to display file available in current directory.

Python's `os` module provides a function that gets a list of files or folders in a directory. The `(.)`, which is passed as an argument to `os.listdir()`, signifies the current folder.

Code:-

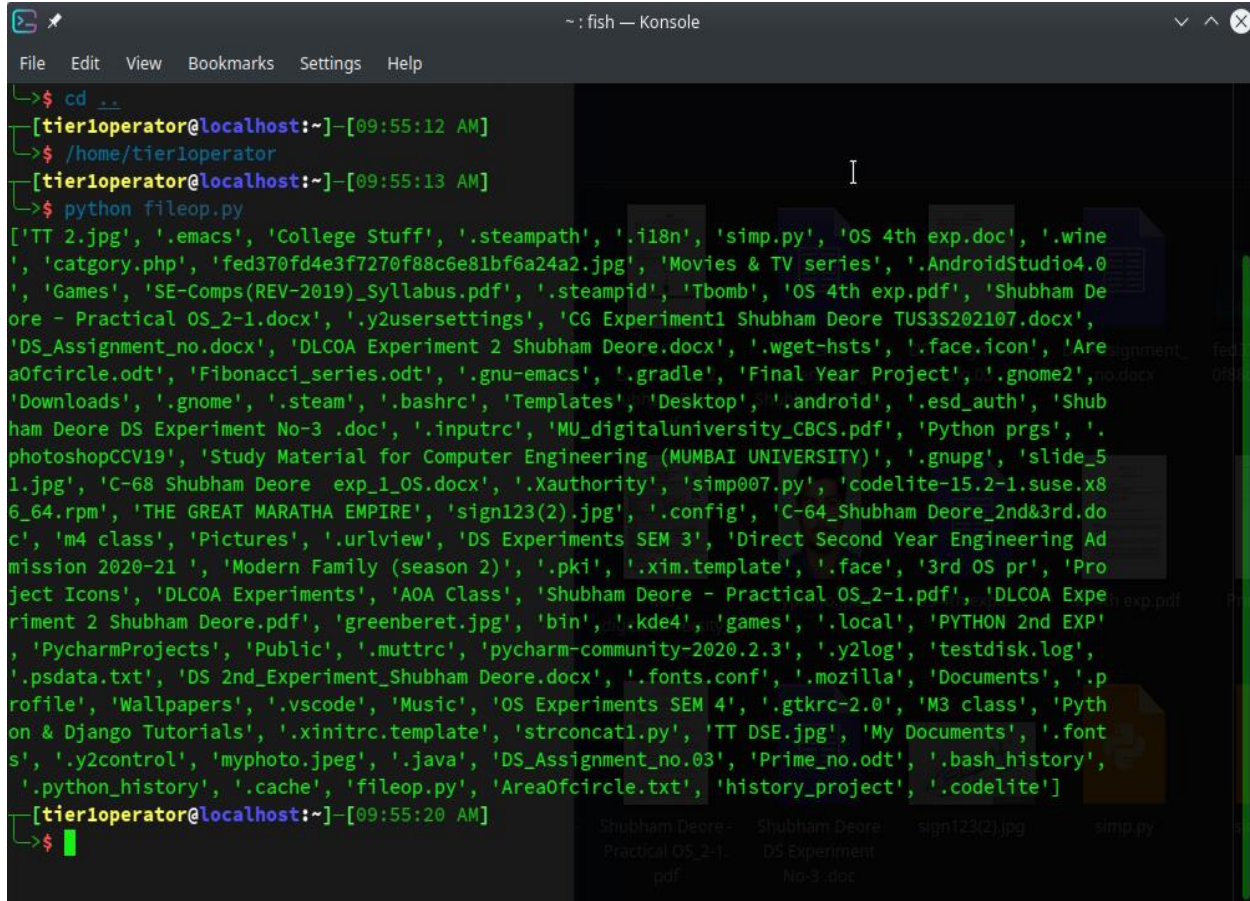
```
import os
```

```
arr = os.listdir('.')

```

```
print(arr)
```

Output :-



```
~: fish — Konsole
File Edit View Bookmarks Settings Help
->$ cd ..
[tier1operator@localhost:~]-[09:55:12 AM]
->$ /home/tier1operator
[tier1operator@localhost:~]-[09:55:13 AM]
->$ python fileop.py
['TT 2.jpg', '.emacs', 'College Stuff', '.steampath', '.i18n', 'simp.py', 'OS 4th exp.doc', '.wine',
'category.php', 'fed370fd4e3f7270f88c6e81bf6a24a2.jpg', 'Movies & TV series', '.AndroidStudio4.0',
'Games', 'SE-Comps(REV-2019)_Syllabus.pdf', '.steampid', 'Tbomb', 'OS 4th exp.pdf', 'Shubham Deore - Practical OS_2-1.docx', '.y2usersettings', 'CG Experiment1 Shubham Deore TUS3S202107.docx',
'DS_Assignment_no.docx', 'DLCOA Experiment 2 Shubham Deore.docx', '.wget-hsts', '.face.icon', 'AreaOfCircle.odt', 'Fibonacci_series.odt', '.gnu-emacs', '.gradle', 'Final Year Project', '.gnome2', 'Downloads', '.gnome', '.steam', '.bashrc', 'Templates', 'Desktop', '.android', '.esd_auth', 'Shubham Deore DS Experiment No-3 .doc', '.inputrc', 'MU_digitaluniversityCBCS.pdf', 'Python prgs', '.photoshopCCV19', 'Study Material for Computer Engineering (MUMBAI UNIVERSITY)', '.gnupg', 'slide_51.jpg', 'C-68 Shubham Deore exp_1_OS.docx', '.Xauthority', 'simp007.py', 'codelite-15.2-1.suse.x86_64.rpm', 'THE GREAT MARATHA EMPIRE', 'sign123(2).jpg', '.config', 'C-64_Shubham Deore_2nd&3rd.doc', 'm4 class', 'Pictures', '.urlview', 'DS Experiments SEM 3', 'Direct Second Year Engineering Admission 2020-21', 'Modern Family (season 2)', '.pki', '.xim.template', '.face', '3rd OS pr', 'Project Icons', 'DLCOA Experiments', 'AOA Class', 'Shubham Deore - Practical OS_2-1.pdf', 'DLCOA Experiment 2 Shubham Deore.pdf', 'greenberet.jpg', 'bin', '.kde4', 'games', '.local', 'PYTHON 2nd EXP', 'PycharmProjects', 'Public', '.muttrc', 'pycharm-community-2020.2.3', '.y2log', 'testdisk.log', '.psdata.txt', 'DS 2nd_Experiment_Shubham Deore.docx', '.fonts.conf', '.mozilla', 'Documents', '.profile', 'Wallpapers', '.vscode', 'Music', 'OS Experiments SEM 4', '.gtkrc-2.0', 'M3 class', 'Python & Django Tutorials', '.xinitrc.template', 'strconcat1.py', 'TT DSE.jpg', 'My Documents', '.font', '.y2control', 'myphoto.jpeg', '.java', 'DS_Assignment_no.03', 'Prime_no.odt', '.bash_history', '.python_history', '.cache', 'fileop.py', 'AreaOfCircle.txt', 'history_project', '.codelite']
[tier1operator@localhost:~]-[09:55:20 AM]
->$
```

B.3 Observations and learning:

Learned to explore file and directories learned about various attributes of files and directories to perform operations on it

B.4 Conclusion:

Able to implement

To explore Files and directories

- Python program to append data to existing file and then display the entire file
- Python program to count number of lines, words and characters in a file.
- Python program to display file available in current directory

B.5 Question of Curiosity

Q.1] Write Different File commands?

The `close()` method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.

The `remove()` Method

You can use the `remove()` method to delete files by supplying the name of the file to be deleted as the argument.

The `rename()` Method

The `rename()` method takes two arguments, the current filename and the new filename

The `write()` Method

The `write()` method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The `rmdir()` Method

The `rmdir()` method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Q.2] Discuss Different File Operations?

Opening a file

To open a file in read or write mode use the built-in `open()` function. This function returns a file object, called a handle which can be used to read or modify the file.

Syntax

```
file_object = open("filename", "mode")
```

Closing a file

When we are done performing the operations on the file, we need to close the file properly. Closing a file will help to free up the resources that were tied up to the file. The built-in `close()` method in Python is used for this operation

Writing to a file

To write into a file in Python, we need to open it in write `w`, append `a` or exclusive creation `x` mode. The `write()` or `writelines()` method is used to write a string or a sequence of bytes(for binary files).

Reading a file

To read a file we have to open the file in reading `r` mode. There are three ways in which the files can be read.

1. `read([n])`
2. `readline([n])`
3. `readlines()`

Append file:

To append to a file we must open the file in `a+` mode.
