

| **TITLE: File handling in Python** |
| --- |

**AIM:** Write a program to demonstrate File handling mechanism in Python

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected OUTCOME of Experiment:** To demonstrate File Handling in python

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Resource Needed: Python IDE**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Theory:**

**Python File Handling**

Till now, we were taking the input from the console and writing it back to the console to interact with the user.

Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console since the memory is volatile, it is impossible to recover the programmatically generated data again and again.

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.

**File operation can be done in the following order:**

Open a file

Read or write - Performing operation

Close the file

**Opening a file:**

Python provides an open() function that accepts two arguments, file name and access mode in which the file is accessed. The function returns a file object which can be used to perform various operations like reading, writing, etc.

**Syntax:**

file object = open(<file-name>, <access-mode>, <buffering>)

The files can be accessed using various modes like read, write, or append. The following are the details about the access mode to open a file.

| **SN** | **Access 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:**

#opens the file file.txt in read mode

fileptr = open("file.txt","r")

if fileptr:

print("file is opened successfully")

**Output:**

<class '\_io.TextIOWrapper'>

file is opened successfullyProblem Defination:

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

**The close() method:**

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.

The syntax to use the close() method is given below.

**Syntax:**

fileobject.close()

**Example:**

try:

fileptr = open("file.txt")

# perform file operations

finally:

fileptr.close()

**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)

**Example:**

# Open a file

fo = open("foo.txt", "wb")

fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opend 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.

**Output:**

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:**

# Open a file

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

str = fo.read(10);

print "Read String is : ", str

# Close opend file

fo.close()

**Output**

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:**

# 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 opend file

fo.close()

**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 −

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 −

import os

# Delete file test2.txt

os.remove("text2.txt")

**Problem Definition**:

Write a program to create a file employeedetails.txt"which stores the Employee details by adding their Employee Id,Name and Department into it using following format:

EmpId Name Department

1601001 Abc Computer

1601003 Xyz IT

Obtain the details for EmpId 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 Programming: A modular Approach, Pearson India, Second Edition 2018,India

**Implementation details:**

a = open("employeedetails.txt", "w")

n = int(input("Enter the number of entries: "))

a.write("EmpId\tName\tDepartment\n")

for i in range(n):

empid = input("Enter EmpId: ")

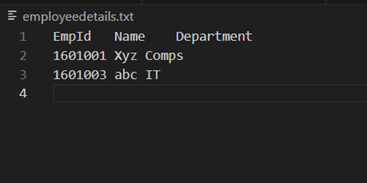
name = input("Enter Name: ")

dep = input("Enter Department: ")

a.write(f"{empid}\t{name}\t{dep}\n")

a.close()

**Output(s):**



**Conclusion:**

Thus, we learned about file handling in python.

**Post Lab 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.
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?

1.

file\_name = input("Enter the file name: ")

try:

with open(file\_name, 'r') as file:

contents = file.read()

print(contents.upper())

except FileNotFoundError:

print("File not found.")

except IOError:

print("Error reading the file.")

2.

It is advised to close a file after we are done with read and write operations for a few reasons:

* Resource Management: When you open a file, the operating system assigns certain resources to handle the file operations. If you don't close the file properly, those resources may not be released until the program terminates, potentially leading to resource leaks. Closing the file ensures that the resources are freed up promptly, allowing other processes or parts of your program to access the file.
* Data Integrity: When you write data to a file, it is usually buffered in memory before being physically written to the disk. Closing the file ensures that any pending data is written to the disk before the file is closed. If you don't close the file, there's a possibility that some data may not be written, leading to data loss or corruption.
* File Locking: In some operating systems, when a file is open, it may be locked to prevent other processes from modifying it concurrently. By closing the file, you release the lock, allowing other processes to access the file if needed.

If you don't close a file after read and write operations, your program may still run without any immediate error messages or warnings. However, as mentioned earlier, it can lead to resource leaks and potential data integrity issues. It's generally considered a good practice to close files explicitly when you're done using them.

**Date: 26/5/23 Signature of faculty in-charge**