Ch-4 - Data File Handling

→ **Transient Program** - Run for short time & produce output, but data clears after the system is turned off because the data entered is executed inside primary memory (RAM) which is volatile (temporary) in nature.

→ **Persient Program** - Run for long time & keep atleast some of the data in permanent storage (eg- hard drive) & if system is shut down or restart, they execute from same point. It is involatile (permanent) in nature.

- Operating systems are persistent programs.
- Simplest way for programs to maintain data are in form of FILES.

Eg- Programs in form of files are used in performing business operations -
Payroll program → keep data about employee in files
Inventory program → keep data about company products in file.
Accounting system → keep data about financial operations in file.

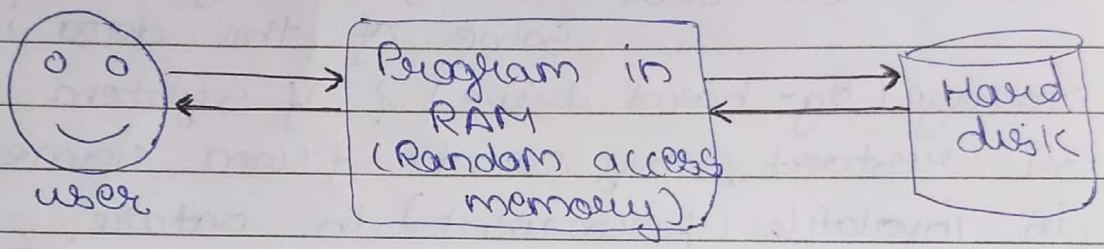
Files - file is a document or data stored on permanent storage device which can be read, written or rewritten acc to requirement.

All files are assigned a name that is used for

identification purpose by operating system and user.

File I/O (Input - Output) - Transferring of data from secondary memory (hard disk) to main memory or vice - versa.

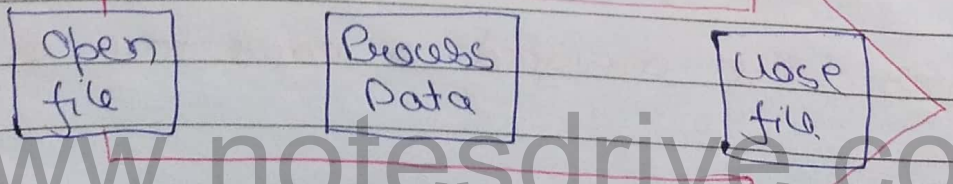
Eg -



- Data maintained inside the files are persister data.
- Python allows us to read or store data to external file permanently and we also stores the program data as Python scripts (with .py extension)
- files provide means of communication b/w the program and outside world.

Python file handling takes place in following order.

1) opening a file
2) performing operations / Processing Data
3) close the file.



File Types

1. **Text Files** - A text file consists of sequence of lines. A line is a sequence of character (ASCII or UNICODE) stored on permanent storage media.

In Text file, each line terminates by special character called End of line (EOL). This EOL character is newline character (\n).

2. **Binary Files** - Files used to store Binary data such as images, video files, audio files etc. It consists of arbitrary binary data, usually numbers stored in file which can be used for numeric operations.

In Binary files, there is no delimiter for a line. Also, no character translation can be carried out in binary files.

- Binary files are easier and much faster than text files for carrying out reading & writing operations on data.

Open() function - It takes the name of file as first argument and file mode as second argument.

Syntax -

```
open (file_name, access_mode)
```

Modes for opening a file -

- read(r) : to read the file
- write(w) : to write to the file
- append(a) : to write at the end of the file

When we work with file(s), a buffer (area in memory where data is temporarily stored before being written to file) is automatically associated with the file when we open it.

While writing the content ~~of~~ to a file, first it goes to buffer, and once the buffer is full the data is written to file.

When the file is closed, any unsaved data is transfered to file.

flush() function is used to force transfer of data from buffer to file.

If opening is successful, the OS return to file handle. File handle is not actual data contained in file; it is used to read the data.

If we do not mention the mode then the file opens in read mode by default.

If the file doesnot exist, open() will fails with a trace back and we will not get a

handle to access the contents of file

r mode - opens a file for reading only.
- This is the default mode
- If file does not exist, it will generate FileNotFoundError
- file pointer is placed in beginning

w mode - opens a file for writing only.
- Overwrite the file if file exist.
- If file donot exist, it will generate a new file for writing.

a mode - opens a file for appending.
- file pointer is at the end of file if file exist.
- If file donot exist, it will create a new file for writing.

.txt extension is used for our convenience of identification as it is easy to identify the file as text file. It is not mendatory to have file name extension

For binary file we will use .dat extension

Another function which can be used for creation of a file is file() Its syntax and its usage is same as open()

Close() function - It flushes any unwritten information and closes the file object, after which no more writing can be done.

Syntax :

file object.close()

Example -

```
>>> f = open("test.txt")
>>> print("The name of file to be closed is", f.na
The name of file to be closed is : test.txt
>>> f.close()
>>>
```

file object

Properties of file Object -

1) name : Name of opened file
2) mode : Mode in which the file get opened
3) Closed : Return boolean value, which indicates whether file is closed or not
4) Readable : Return boolean value which indicates whether file is readable or not

Reading from a file -

1) read() : To read entire data from the file. Start reading from cursor to end of file.

2) Eg -

NOTEPAD: test.txt
Hello user you are working with Python files.

Program

```
f=open("test.txt","r")      # Opening file in read mode.
data = f.read()
print(data)
f.close()
```

Output

```
Hello user
you are working with
python files.
```

2) read(n) : To read 'n' character from file, starting from the cursor; if file contains less than n character than it will read untill the end.

Program

```
f = open("test.txt","r")      # Program to read first 10 characted
data = f.read(10)
print(data)
f.close
```

Output

```
Hello User.
```

c) Readline() - To read only one line from the file; starts reading from cursor up to, and including the end of line character.

first call to function will return first line, second call the next line and so on.

line is terminated by '\n' or end = ' '

Program

```
f = open("test.txt", 'r')      # Program to read data line by line from a file
l1 = f.readline()
print(l1, end=' ')
l2 = f.readline()
print(l2, end=' ')
f.close()
```

Output

```
Hello user
you are working with
>>> 1.
```

d) Readlines() - To read all lines from the file into a list; start reading from cursor and returns a list of lines seperated by \n.

Program

```
f = open("test.txt",'r')          # Program to read
l = f.readlines()                 all the lines into a
for x in l:                       list (using readlines())
    print(x,end=' ')
f.close.
```

Output

```
Hello user.
you are working with
python file.
```

Some more modes

- **wb mode** - Opens a file for writing only in binary format. Overwrites the file if it exist & creates a new file if it donot exist.
- **rb mode** - This opens a file for reading only in binary format. File pointer is placed in the begining of file.
- **ab mode** - Opens a file for appending only in binary format. Pointer is at the end if the file exist and creates a new file if the file doesn't exist.
- **rb+ mode** - Opens the file for both reading and writing ~~format~~ in binary format.

file pointer is placed in the beginning of file.

- wb+ mode - Open the file for both writing and reading in binary format. Overwrite existing files, if the file exist. If file doesnot exist, it will creates a new file.

- ab+ mode - opens the file for both appending and reading in binary format. The file pointer is at the end of file if file exist. The file opens in append mode & if the file doesnot exist it creates a new file for reading and writing.

WRITING TO FILE

We can write data into file by two methods:

1) write (string)
2) writelines (sequence of line).

1) write() - It takes a string as parameter and write its in a file.

- Always add '\n' at the end of line.

Syntax:

fileObject.write(string)

Example -

Program to write data to a file.

Program -

```
f = open("test2.txt","w")
f.write("We are writing \n")
f.write("text file \n")
print f.write("Data written to file successfully")
f.close()
```

output

```
Data written to file successfully.
>>>
```

- For numeric data value, conversion to string is required.

Example -

```
x = 52
file.write(str(x))
```

Practical Implementation

Adding numeric to a file using conversion function - str()

```
f = open("newtest.txt", 'w')
x = 100
f.write("Hello world" \n")
f.write(str(x))
f.close()
```

Output.

>>> 1 → Blinking cursor indicates the successful write operation in the file.

```
f = open("newtest.txt", 'r').
data = f.read()
print(data)
f.close()
```

Output

Hello World

100

Writelines()

Sequence datatype including string can be written using writelines() method in file.

Syntax -

fileobject.writelines(sequence)

Practical implementation

```
# Program to illustrate writelines() method
# for writing list into file

f = open("test4.txt",'w')
f.writelines("Computer Science\n","Physics\n",
             "Chemistry\n","Maths\n")
f.close()
```

Output

```
>>> 1
```

```
f = open("test4.txt",'r')
d = f.read()
print(d)
f.close()
```

Output

```
>>> Computer Science
    Physics
    Chemistry
    Maths
```

With statement

Apart from using open() function for creating a file, with statement can also be used for same purpose.

Using with ensures that all the resources allocated to the file object get deallocated automatically once we stop using the file.

We are not required to close the file explicitly using with statement.

Syntax:

```
with open() as file object :
    file manipulation statements
```

Practical illustration

```
# Program to illustrate 'with' statement

with open ("test1.txt",'w') as f :
    f.write("Python \n")
    f.write("is an easy language \n")
    f.write("to work with \n")
    print("Is file closed:", f.closed)
print("Is file closed :", f.closed)
```

```
>>> Is file closed : False
    Is file closed : None
```

```
with open("test1.txt", 'r') as f:
    d=f.read()
    print(d)
```

```
>>> Python
    is an easy language
    to work with.
```

Appending to file

Open for writing, and if it exists, then append data to the end of file.

Syntax :

```
fileobject = open("filename", 'a')
```

Practical implementation

Program

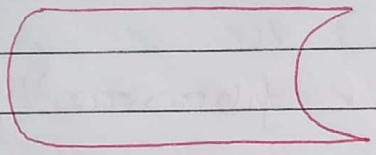
```
# Program to add data in existing files
f=open("test1.txt", 'a')
f.write("~~Programming~~ Simple syntax \n")
f.write("makes python easy")
print("more data appended")
f.close
```

```
>>> More data appended
```

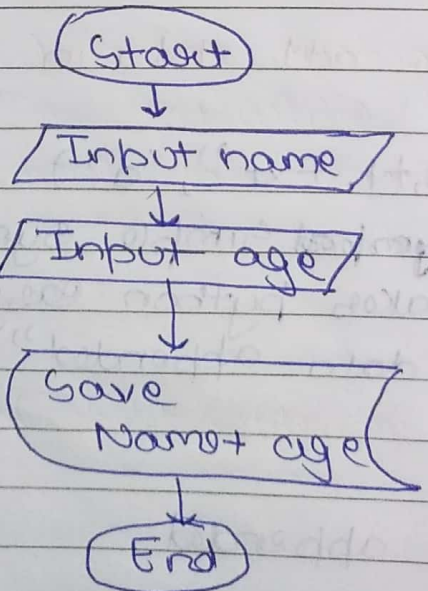
```
f = open("test1.txt", 'r')
d = f.read()
print(d)
f.close()
```

>>> Python
is an easy language
to work with
Simple Syntax
makes python easy.

While making flow charts, we can store data inside a file using a specific symbol



Q) Design a flow chart and a program to ask user to input his name along with age and store data in text file.



Program

```
name = input("Enter name")
age = int(input("Enter age"))
f = open("user.txt", 'a')
f.write(name)
f.write(age)
f.close()
```

Output

```
Enter your name : Harsh
Enter age       : 17

Enter your name : Ankit
Enter age       : 25
>>>
```

To check

```
f = open("user.txt", 'r')
f1 = f.read()
print(f1)
f.close()

>>> Harsh17Ankit25
```

BINARY FILE OPERATIONS

To write a structure such as list or dictionary to a file and read it we need to use pickle.

Pickle - It refers to the process of converting structure to byte stream before writing to file

While reading the content, a reverse process called Unpickling is used to convert byte stream back to original structure.

Pickle module is used for storing data in binary format as it allows us to store python object with their structure.

For importing modules, there are two main methods :

1) Pickle.dump() :- For creation of binary file and to write the object in file, which is opened in binary access mode.

Syntax -

pickle.dump(object, file object)

PRACTICAL IMPLEMENTATION

1) # Program to write list sequence in a binary file.

```
def foperation():
    import pickle
    l = [10, 20, 30, 40, 100]
    f = open ('list.dat', 'wb')
    pickle.dump (l, f) # writing content to binary file
    print ("file added to binary file")
    f.close
foperation()
```

output

file added to binary file

2) # Program to write dictionary to a binary file.

```
import pickle
d = {'Python': 90, 'Java': 95, 'C++': 85}
f = open ('bin_file.dat', 'wb')
pickle.dump (d, f)
f.close()
```

bin_file.dat - NOTEPAD.
€}q{X-Pythonq KZX-Javaq-K_X-C++q-KUu

Dictionary items are added to binary file 'bin_file.dat', but it is not in human readable form.

2) pickle.load() :- once data is stored using dump(), it can be used for reading. for reading data from the file we have to use pickle.load()

Syntax :

object = pickle.load(file object)

* we need to use load() each time dump() is used

Practical Implementation

```
# Program to read python dictionary contents
  back from the file

import pickle
f = open('bin_file.dat','rb')
d = pickle.load(f)      # reading data from
                          binary file
f.close()
print(d)
```

output

{'Python':90, 'Java':95, 'C++':85}

But still in notepad the same unreadable form is found.

Relative And Absolute Paths -

- Files are organised into directories (also called folders).
- Every running program has a 'current directory', which is default directory for most operations.

os module provides functions for working with files and directories.

* os.getcwd return name of current directory
  ↳ current working directory.

```
import os                                  (text file is opened
cwd = os.getcwd()                          through absolute path)
print(cwd)
```

Output

C:\Users\AppData\Local\Programs\Python36

Absolute path.

\\test.txt ← Relative path

- Files are always stored in current folder / directory by default.
- Os module of Python provides various methods to work with file and folder. for using these functions, we have to import os module in our program.

Absolute Path - It is full path points to same location in a file system, regardless of current working directory.

It must include the root directory

Relative Path - Starts from given working directory, avoiding the need to provide full path.

- Python program and external file must be in the same directory, else we will need to enter the entire file path.

Standard File Streams -

Standard streams available in python-

→ Standard input stream
→ Standard output stream
→ Standard error stream.

These streams are nothing but file object, which get automatically connected to your program's standard device when we start python.

In order to work with these streams, we need to import sys module.

Methods which are available for input output operations in it are read() for reading a byte at a time from keyboard, write() for writing data on console (monitor).

There are three standard streams -

1) sys.stdin : When a stream read from standard input.
2) sys.stdout : Data written to sys.stdout typically appears on screen but can be linked to standard input of another program with a pipe.
3) sys.stderr : Error messages are written to sys.stderr.

Practical Implementation :

Program to implement standard streams.

```
import sys
f = open("test.txt", 'r')
l1 = f.readline()
l2 = f.readline()
l3 = f.readline()
sys.stdout.write(l1)
sys.stdout.write(l2)
sys.stdout.write(l3)
```

} write the respective lines on device associated with sys.stdout which is the monitor

Output

```
Hello user
you are working with
Python files
```