FILE HANDLING

TOPICS

1. DRAWBACK OF VARIABLES
   a. Variables are used to store the less amount of data
   b. variables are volatile(temporary) in nature
2. FILE INTRODUCTION
   ⇨ File concept is used when we want to store the large amount of data and to store it permanently
   ⇨ Files are used to store the data permanently in secondary devices like “Hard Disk”
   ⇨ Files are non-volatile(permanent) in nature
   ⇨ Files are used to store the large amount of data
3. TYPES OF FILES

   There are mainly two types of files:

   A. Text Files
   B. Binary Files

   TEXT FILES

   ⇨ These files are used to store the character data
   ⇨ In simple words, these files contain the plain text data
   ⇨ These files can be readable and understandable
   Examples:
   - Html, xml files
   - Source code files such as python, c, c++
   - Document files such as TXT, RTF, TEX
   - Log files such as server logs and application logs
   - Data files such as CSV, JSON
   - Configuration files such as INI, YAML

BINARY FILES

- ⇨ These files are used to store the binary data
- ⇨ In simple words, these files contain binary data such as images, videos, audio files etc
- ⇨ Binary files are larger than the text files
- ⇨ These files can't be readable and understandable because the data which is inside the file encoded in the binary format i.e., 0's and 1's which can be understandable only by the machine
  Examples:
  - Document files such as DOCX, XLS, PDF etc
  - Image files such as PNG, JPEG, GIF etc
  - Video files such as MP4. MOV, AVI etc
  - Audio files such as MP3, AAC, WAV etc
  - Database files such as mdb, accde, sqlite, frm etc
  - Archive files such as ZIP, RAR etc
  - Executable files such as EXE, DLL, etc
- ⇨ If we want to read the data inside the binary files we need specific software's like:
  - For video files – video player
  - For audio files – media player
  - For image files – photo editor
  - For docx files – Microsoft word software

4. FILE OPENING MODES

⇨ File opening modes are used to open the file in specific mode like read, write etc based on our requirement

TEXT FILE OPENING MODES:

- r mode:
  - this mode is used to open the file for reading
  - it is used to read the data from the file
  - file pointer is placed at beginning of the file
  - if reads the data if the file exists
  - if the file doesn't exist, we get "FileNotFoundError"
- w mode:
  - this mode is used to open the file for writing
  - when you open the file in w mode, whatever the data available inside the file will be deleted(truncated)
  - whatever the data you will written that will overrides the existing data
  - if the file doesn't exist, the new file will be created
  - the file pointer placed at the beginning of the file
- a mode:
  - this mode is used to open the file for appending
  - in simple words, it is used to write the data to the end of the file
  - if the file doesn't exist, the new file will be created and the file pointer will be at the beginning of the file
  - if the file exists, the file pointer will be at the end of the file

- x-Exclusive creation mode: fails if the file already exists
- other modes like:
  - r+ -- read and write
  - w+ -- write and read
  - a+ -- append and read

BINARY FILE OPENING MODES

  - rb – for reading
  - wb – for writing
  - ab – for appending
  - rb+ -- read and write
  - wb+ -- write and read
  - ab+ -- append and read

5. FILE PROPERTIES
   a. name: it returns the name of the file
   b. mode: it returns the mode of the file
   c. readable():
      i. it checks whether the file is readable or not
      ii. returntype is Boolean
   d. writable():
      i. it checks whether the file is readable or not
      ii. returntype is Boolean
   e. closed:
      i. it checks whether the file is closed or not
      ii. returntype is Boolean

ex:

```
fileobj=open("samplefile","r")
print(fileobj.name)
print(fileobj.mode)
print(fileobj.readable())
print(fileobj.writable())
print(fileobj.closed)
fileobj.close()
print(fileobj.closed)
```

6. SEEK() AND TELL() METHODS

seek()

- ⇨ This function is used to move the file pointet/cursor to the specific location
- ⇨ Syntax:

```
seek(offset,fromwhere)
```

  - values:
    0 – from beginning
    1 – from the current position
    2 – to the end
- ⇨ but python supports only one value i.e., 0 – from the beginning

tell()

- ⇨ this function returns the current location of the file pointer/cursor

7. FILE HANDLING OPERATIONS

⇨ Before we perform any operations on the file, first we need to open the file

⇨ We must close the file after the operation is completed because whenever we open the file then we get file is already in read mode, already in write mode etc

A. OPENING A FILE

a. We can open the file by using open() function which is the built in function

b. Syntax:

```
fileobj=open(filename,mode)
```

- filename – it is the name of the file which you want to open
- mode – it is used to specify the mode in which file should be opened
- this function returns the fileobject which is used to access read, write and other built in fuctions

ex:

```
fileobj=open("samplefile","r")
```

- OPENING A FILE BY USING "WITH"
- we can also open the file by using "with"
- the main advantage by using with:
  - we don't need to close the file manually
  - when we come out of the with block automatically the file will be close
- syntax:

```
with open(filename,mode) as fileobj:
        operations
```

ex:

```
with open("samplefile","w") as fileobj:
    fileobj.write("iam c++")
    print(fileobj.closed)
print(fileobj.closed)
```

B. READING THE DATA FROM A FILE

We can read the data from the file by using 3 methods:

1. read():
   a. this method is used to read the entire data at a time from the file
   b. syntax:

```
fileobj.read()
```

   c. it will return the entire data as a string format
   d. we can also use this method like read(n)
      i. n – we can mention how many characters to read
   e. when you perform the read operations the file pointer moves forward so when you perform the read operation again the reading operation starts from where the previous read operation ended so in this case we use seek() method to read from the beginning

   ex:

```
fileobj=open("samplefile","r")
data=fileobj.read()
print(data)
fileobj.seek(0)
data2=fileobj.read(6)
print(data2)
```

2. readline()
    a. this method is used to read the single line from the file
    b. syntax:

```
fileobj.readline()
```

    c. we can also use this method like readline(n)
        i. n – we can mention how many characters to read from a single line
    d. it will return the single line as a string format

    ex:

```
fileobj=open("samplefile","r")
data=fileobj.readline()
print(data)
fileobj.seek(0)
data2=fileobj.readline(7)
print(data2)
```

3. readlines()
    a. this method is used to read all the lines from the file
    b. it returns the list of strings,
        i. each string in the list == each line in the file
    c. syntax:

```
fileobj.readlines()
```

    ex:

```
fileobj=open("samplefile","r")
li=fileobj.readlines()
for i in li:
    print(i)
```

C. WRITING THE DATA TO THE FILE

We can write the data to the file in 2 ways:

1. write()
   a. this method is used to write only one line to the file
   b. syntax:

```
fileobj.write(string)
```

   ex:

```
fileobj=open("samplefile","w")
fileobj.write("iam c++")
fileobj.close()
```

2. writelines()
   a. this method is used to write the multiple lines to the file by passing list of strings to the method
   b. each string is considered as a one line in the list
   c. syntax:

```
fileobj.writelines([string1,string2,..])
```

   ex:

```
fileobj=open("samplefile","w")
fileobj.writelines(["iam python\n","iam
java\n","iam c"])
fileobj.close()
```

D. APPENDING DATA TO THE FILE

i. We can write the data to the end of the file by opening the file in append mode

ii. syntax:

```
fileobj.write(string)
```

ex:

```
fileobj=open("samplefile","a")
fileobj.write("\niam c++")
fileobj.close()
```

E. CLOSING A FILE

⇨ We must close the file after performing any operation to the file

⇨ Especially we must close the file after the write() method because if you don't close whatever the data that you have written it will not be saved to the file

⇨ Syntax:

```
fileobj.close()
```

F. RENAMING A FILE

⇨ We can rename the file by using rename() method

⇨ rename() is the built in method which is available in "os" module

⇨ syntax:

```
os.rename(old_name,new_name)
```

ex:

```
import os
os.rename("samplefile","sample1file")
```

G.DELETING A FILE

- ⇨ We can delete the file by using remove() method
- ⇨ remove() is the built in method which is available in "os" module
- ⇨ syntax:

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

ex:

```
import os
os.remove("sample1file")
```

H. DELETING FILES AND FOLDERS

- ⇨ To remove the single file we use remove()
- ⇨ To remove the empty directory we use rmdir()
- ⇨ To remove the directory and all its contents we use rmtree()
- ⇨ remove() and rmdir() methods are available in "os" module
- ⇨ rmtree() method is available under "shutil" module

TO REMOVE A SINGLE FILE

Ex:

```
import os
filename=input("enter the file name")
if os.path.exists(filename):
    print("file is found")
    os.remove(filename)
    print("file is removed successfully")
else:
    print("file is not found")
```

DELETING AN EMPTY FOLDER

⇨ before removing a file it must be empty, because python checks for the file whether it is empty or not

⇨ it will show the warning if it is not empty

Ex:

```
import os
folder_name=input("enter the folder name")
if os.path.exists(folder_name):
    print("folder is found")
    if len(os.listdir(folder_name))==0:
        print("folder is found empty")
        os.rmdir(folder_name)
        print("folder is removed successfully")
    else:
        print("folder is not found")
else:
    print("folder is not found ")
```

DELETING A FOLDER AND ALL ITS CONTENTS

Ex:

```
import shutil
import os
folder_name=input("enter the folder name")
if os.path.exists(folder_name):
    print("folder is found")
    shutil.rmtree(folder_name)
    print("entire folder is succesfully deleted")
else:
    print("folder is not found")
```