FILE HANDLING

Problem

- Consider the following scenario

  -> You are asked to find the number of students who secured centum in mathematics in their examination. A total of 6 lakhs students appeared for the examinations and their results are available with us.

- The processing is typically supposed to be automatic and running on a computer. As data are most helpful when presented systematically and in fact educational to highlight their practicality.

```
Pseudocode
OPEN file
REPEAT
    READ each line of file
    SET count = 0
    PARSE the line
    IF maths_mark == 100 THEN
            COMPUTE count as count + 1
    END IF
until end of file is reached
PRINT count
```

Necessity of Files

- Small businesses accumulate various types of data, such as financial information related to revenues and expenses and data about employees, customers and vendors.
- Traditional file organization describes storing data in paper files, within folders and filing cabinets.
- Electronic file organization is a common alternative to paper filing; each system has its benefits and drawbacks.

What is a file?

- A file is some information or data which stays??? in the computer storage devices. We already know about different kinds of file, like music files, video files, text files, etc.

Introduction to file handling

- Files – Huge volume or Collection of data
- Types – Binary, Raw, Text, etc.
- Open any file before read/write.

Opening a File

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

Modes of opening a File:

- r – Reading only
- r+ - Both Read/Write
- w – Writing only
- w+ - Both Read/Write
- a – Appending
- a+ - Appending/Reading

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).

Functions for file Handling

- The read functions contains different methods:
  - read() #return one big string
  - readline() #return one line at a time
  - readlines() #returns a list of lines

Example – read()

welcome.txt
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!

```
f = open(“a1.py", "r")
print(f.read())

f = open("D:\\myfiles\welcome.txt", "r")
print(f.read())

F=open(“f:\\cauverynews.txt”,”a”)
```

- By default the read() method returns the whole text, but you can also specify how many characters you want to return:

```
f = open(“file1.txt", "r")
print(f.read(5))
```

readline()

- the readline() method return one line :

```
f = open(“file1.txt", "r")
    print(f.readline())
```

- By calling readline() two times, you can read the first two lines:

```
f = open(“file1.txt", "r")
print(f.readline())
print(f.readline())
```

readlines()

- Return all lines in the file, as a list where each line is an item in the list object:

```
f=open("D:\\test1.txt")
print(f.readlines())
```

['Hello! Welcome to demofile.txt \n', 'This file is for testing purposes.\n', 'Good Luck!\n']

- Loop through the file

```
f = open(“file1.txt", "r")
for x in f:
        print(x)
```

- It is a good practice to always close the file when you are done with it.

```
f = open(“file1.txt", "r")
print(f.readline())
f.close()
```

File Object Attributes

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.

Example

```
f=open("D:\\test1.txt“)
print("Name of the file:",f.name)
print("Closed or not:",f.closed)
print("Opening mode:",f.mode)
```

```
Name of the file: D:\test1.txt
Closed or not: False
Opening mode: r
```

File Handling functions contd..

- This method writes a sequence of strings to the file.
  - write () #Used to write a fixed sequence of characters to a file
  - writelines() #writelines can write a list of strings.

Append()

- The append function is used to append to the file instead of overwriting it.
- To append to an existing file, simply open the file in append mode ("a"):
- When you’re done with a file, use close() to close it and free up any system resources taken up by the open file.

Write to an Existing File

- To write to an existing file, you must add a parameter to the open() function:
- "a" - Append - will append to the end of the file
- "w" - Write - will overwrite any existing content

Examples:

- Open the file “file2.txt" and append content to the file:

```
f = open(“file2.txt", "a")
f.write("Now the file has more content!")
f.close()
#open and read the file after the appending:
f = open(“file2.txt", "r")
print(f.read())
```

- Open the file “file3.txt" and overwrite the content:

```
f = open(“file3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the appending:
f = open(“file3.txt", "r")
print(f.read())
# the "w" method will overwrite the entire file.
```

writelines()

- Open the file with "a" for appending, then add a list of texts to append to the file:

```
f = open(“file3.txt", "a")
f.writelines([“See you soon!", "Over and out."])
f.close()
#open and read the file after the appending:
f = open(“file3.txt", "r")
print(f.read())
```

Hello! Welcome to demofile2.txt
This file is for testing purposes.
Good Luck!See you soon!Over and out.

To open a text file, use:

```
fh = open("hello.txt", "r")
```

To read a text file, use:

```
fh = open("hello.txt","r")
print (fh.read() )
```

To read one line at a time, use:

```
fh = open("hello".txt", "r")
print (fh.readline() )
```

To read a list of lines use:

```
fh = open("hello.txt.", "r")
print (fh.readlines() )
```

To write to a file, use:

```
fh = open("hello.txt","w")
fh.write("Hello World")
fh.close()
```

To write to a file, use:

```
fh = open("hello.txt", "w")
lines_of_text = ["a line of text", "another line of text", "a third line"]
fh.writelines(lines_of_text)
fh.close()
```

To append to file, use:

```
fh = open("Hello.txt", "a")
write("Hello World again")
fh.close ()
```

Playing Randomly in files

- fileObject.tell() -> current position within a file
- fileObject.seek(offset [,from]) -> Move to new file position.
  - Argument offset is a byte count.
  - Optional argument whence defaults to 0 (offset from start of file, offset should be >= 0); other values are 1 (move relative to current position, positive or negative), and 2 (move relative to end of file, usually negative, although many platforms allow seeking beyond the end of a file)

Sample Reading and Writing

```
fh = open('hello.txt', 'w+')
fh.write("Hello World")
fh.close()
fh = open('hello.txt', 'r+')
print (fh.read())
fh = open("Hello.txt", "a")
fh.write("Hello World again")
fh.close ()
fh = open('hello.txt', 'r+')
print (fh.read())
```

Example for random seeking

```
fo = open("hello.txt", "r+")
str = fo.read(10);
print("Read String is : ", str)

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

# Reposition pointer to start from 10
position = fo.seek(10);
str = fo.read(20);
print("Again read String is : ", str)
# Close opend file
fo.close()
```

seek()

- Change the current file position to 4, and return the rest of the line:

```
f = open("demofile.txt", "r")
f.seek(4)
print(f.readline())
```

```
o! Welcome to demofile.txt
```

tell()

- The tell() method returns the current file position in a file stream.
- Find the current file position:

```
f = open("demofile.txt", "r")
print(f.tell())
```
0

```
f = open("demofile.txt", "r")
print(f.readline())
print(f.tell())
```
30

Tips and Tricks makes it Easier…

- Number of characters in a file is same as the length of its contents.

```
def charcount(filename):
    return len(open(filename).read())
```

- Number of words in a file can be found by splitting the contents of the file.

```
def wordcount(filename):
    return len(open(filename).read().split())
```

- Number of lines in a file can be found from readlines method.

```
def linecount(filename):
    return len(open(filename).readlines())
```