Computer Science (083)
CLASS-XII 2021-22

DISTRIBUTION OF MARKS:

UNIT	UNIT NAME	MARKS
I	Computational Thinking and Programming - 2	40
II	Computer Networks	10
III	Database Management	20
TOTAL		70

Unit No	Unit Name	Term-1	Term-2
I	Computational Thinking and Programming - 2	35	5
II	Computer Networks	---	10
III	Database Management	---	20
	Total	35	35

TERM 1:
Unit I: Computational Thinking and Programming – 2

- Revision of Python topics covered in Class XI.
- Functions: types of function (built-in functions, functions defined in module, user defined functions), creating user defined function, arguments and parameters, default parameters, positional parameters, function returning value(s), flow of execution, scope of a variable (global scope, local scope)
- Introduction to files, types of files (Text file, Binary file, CSV file), relative and absolute paths
- Text file: opening a text file, text file open modes (r, r+, w, w+, a, a+), closing a text file, opening a file using with clause,
- Binary file: basic operations on a binary file: open using file open modes (rb, rb+, wb, wb+, ab, ab+), close a binary file, import pickle module, dump() and load() method, read, write/create, search, append and update operations in a binary file
- CSV file: import csv module, open / close csv file, write into a csv file using csv.writerow() and read from a csv file using csv.reader( )

File Handling:

A file is a sequence of bytes on the disk/permanent storage where a group of related data is stored. File is created for permanent storage of data.
In programming, Sometimes, it is not enough to only display the data on the console. Those data are to be retrieved later on, and then the concept of file handling comes. It is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the file system which is not volatile and can be accessed every time. Here, comes the need of file handling in Python.

File Handling is a mechanism by which we can read data of disk files in python program or write back data from python program to disk files.
File handling in Python enables us to create, update, read, and delete the files stored on the file system through our python program. The following operations can be performed on a file.

Types of File:

There are two types of files:

Text Files- A files whose contents can be viewed using a text editor is called a text file. A text file is simply a sequence of ASCII or Unicode characters. Python programs, contents written in text editors are some of the example of text files. e.g. .txt,.rtf,.csv etc.

Binary Files-A binary file stores the data in the same way as stored in the memory. The .exe files, mp3 file, image files, word documents are some of the examples of binary files. We can't read a binary file using a text editor .e.g. .bmp, .cdr etc.

CSV File- CSV (Comma Separated Values) is a file format for data storage which looks like a text file. The information is organized with one record on each line and each field is separated by comma.

Text File	Binary File
Its Bits represent character.	Its Bits represent a custom data.
Less prone to get corrupt as change reflects as soon as made and can be undone.	Can easily get corrupted, corrupt on even single bit change
Store only plain text in a file.	Can store different types of data (audio, text, image) in a single file.
Widely used file format and can be opened in any text editor.	Developed for an application and can be opened in that application only.
Mostly .txt,.rtf are used as extensions to text files.	Can have any application defined extension.

Text File Handling

In Python, File Handling consists of following three steps:

Open the file.
Process file i.e perform read or write operation.
Close the file.

Opening a File:

File can be opened for either – read, write, append.

SYNTAX:

file_object = open(filename)

Or

file_object = open(filename,mode,buffer)

Filename = name of the file , enclosed in double quotes.
Mode = Determines the what kind of operations can be performed with file like read, write etc.

Buffer = for no buffering set it to 0.for line buffering set it to 1.if it is greater than 1 ,then it is buffer size. if it is negative then buffer size is system default.

Text File Mode	Binary File Mode	Description	Notes
'r'	'rb'	Read only	File must exists, otherwise Python raises I/O errors
'w'	'wb'	Write only	If file not exists, file is created If file exists, python will truncate existing data and overwrite the file.
'a'	'ab'	Append	File is in write mode only, new data will be added to the end of existing data i.e. no overwriting. If file not exists it is created
'r+'	'r+b' or 'rb+'	Read and write	File must exists otherwise error is raised Both reading and writing can take place
w+	'w+b' or 'wb+'	Write and read	File is created if not exists, if exists data will be truncated, both read and write allowed
'a+'	'a+b' or 'ab+'	Write and read	Same as above but previous content will be retained and both read and write.

Example:

myfile = open("story.txt")

here disk file "story.txt" is loaded in memory and its reference is linked to "myfile" object, now python program will access "story.txt" through "myfile" object.

here "story.txt" is present in the same folder where .py file is stored otherwise if disk file to work is in another folder we have to give full path.

Example:

myfile = open("test.txt","r")

here "r" is for read (although it is by default, other options are "w" for write, "a" for append)

Example:

myfile = open("d:\\mydata\\poem.txt","r")

here we are accessing "poem.txt" file stored in separate location i.e. d:\mydata folder.

at the time of giving path of file we must use double backslash (\\) in place of single backslash because in python single slash is used for escape character and it may cause problem like if the folder name is "temp" and we

provide path as d:\temp\poem.txt then in \temp "\n" will become escape character for new line, so always use double backslash in path

Another solution of double backslash is using "r" before the path making the string as raw string i.e. no special meaning attached to any character as:

myfile = open(r"d:\mydata\poem.txt","r")

In the above example "myfile" is the file object or file handle or file pointer holding the reference of disk file.

Closing a File:

- As reference of disk file is stored in file handle so to close we must call the close() function through the file handle and release the file.

myfile.close()

Note: open function is built-in function used standalone while close() must be called through file handle

Reading from a File:

To read from file python provide many functions like:

FileObject.read([n]) : reads and return n bytes, if n is not specified it reads entire file.
FileObject.readline([n]) : reads a line of input. If n is specified reads at most n bytes. Read bytes in the form of string ending with line character or blank string if no more bytes are left for reading.

FileObject.readlines(): reads all lines and returns them in a list

Example (raed()):

Test.txt

```
this is my first file
this is first line
this is second line
this is third line
```

```
# data file handling to read
myfile = open('test.txt', 'r')
str1 = myfile.read(10)
```

```
str2 = myfile.read(5)
str3 = mtfile.read()
print(‘first read() will show =’,str1)
print(‘second read() will show =’,str2)
print(‘third read() will show =’,str3 )

output:
first read() will show = this is my
second read() will show =  firs
third read() will show = t file
this is first line
this is second line
this is third line
```

```
# data file handling to read
myfile = open('test.txt','r')
str1 = myfile.read()
str2 = myfile.read(10)
str3 = myfile.read(5)

print('first read() will show =',str1)
print('second read() will show =',str2)
print('third read() will show =',str3 )

Output:
first read() will show = this is my first file
this is first line
this is second line
this is third line

second read() will show =
third read() will show =
```

Example (readline())

Test.txt

```
this is my first file
this is first line
this is second line
this is third line
```

```
# data file handling to readline
myfile = open('test.txt','r')
```

```
line1 = myfile.readline()
line2 = myfile.readline()
line3 = myfile.readline(10)
print(line1)
print(line2)
print(line3)

Output:
this is my first file

this is first line

this is se
```

```
# data file handling to readline
myfile = open('test.txt','r')
line1 = myfile.readline()
line2 = myfile.readline()
line3 = myfile.readline(10)
print(line1,end='')
print(line2,end='')
print(line3)

Output:
this is my first file
this is first line
this is se
```

Example (read line by line)

Test.txt

```
this is my first file
this is first line
this is second line
this is third line
# data file handling to read line by line
myfile= open('test.txt', 'r')
str=' '
while str:
   str = myfile.readline()
   print(str,end='') # diff between '' and ' '
myfile.close()
Output:
this is my first file
```

```
this is first line
this is second line
this is third line
```

```
# data file handling to read line by line
myfile= open('test.txt', 'r')
for str in myfile:
   print(str,end='') # diff between '' and ' '
myfile.close()

Output:
this is my first file
this is first line
this is second line
this is third line
```

```
myfile= open('test.txt', 'r')
str1= ‘ ‘
size=0
tsize=0
while str1:
        str1=myfile.readline()
        tsize=tsize+len(str1)
        size=size+len(str1.strip())
print(“Total Size=”, tsize)
print(“Size after removing EOL and blank”, size)
myfile.close()

Output:
Total Size= 81
Size after removing EOL and blank 76
```

Example readlines()

Test.txt

```
this is my first file
this is first line
this is second line
this is third line
```

```
myfile= open('test.txt', 'r')
contents = myflie.readlines()
```

```
print(contents)
print(“First line is:”, contents[0])
print(“Last line is:”, contents[len(contetns)-1])

Output:
['this is my first file\n', 'this is first line\n', 'this is second line\n', 'this is third line']
First line is: this is my first file

Last line is: this is third line
```

More Examples:

```
myfile =open('test.txt','r')
str=myfile.read()
size=len(str)
print('Size of file in bytes=', size)
myfile.close()

Output;
Size of file in bytes= 79
```

```
myfile =open('test.txt','r')
str=myfile.readlines()
lines=len(str)
print('Number of lines in file=', lines)
myfile.close()

Output;
Number of lines in file= 4
```

Writing on to Files:

After read operation, let us take an example of how to write data in disk files. Python provides functions:

FileObject.write (str1) - Writes string str1 to file referenced by file handle

FileObject.writelines (L) - Writes all string in List L as lines to file referenced by file handle.

The above functions are called by the file object to write desired content.

Example: write()

```
myfile=open('student.txt','w')
for i in range(3):
   name = input('Enter name to store :')
   myfile.write(name)
   myfile.write('\n')
myfile.close()
print('\n data saved successfully...')
```

Example: append

```
myfile=open('student.txt','a')
for i in range(3):
   name = input('Enter name to store :')
   myfile.write(name)
   myfile.write('\n')
myfile.close()
print('\n data saved successfully...')
```

Example: writelines()

```
myfile=open('emp.txt','w')
mylist = []
for i in range(3):
   name = input('Enter name of employee :')
   mylist.append(name+'\n')

myfile.writelines(mylist)
myfile.close()
print('\n data saved successfully...')
```

Example: Writing String as a record to file

```
myfile = open ('book.txt', 'a')
ans='y'
while ans == 'y':
   bno = int(input('Enter Book Number:'))
   bname = input('Enter Book Name:')
   author = input('Enter Author Name:')
   price = int(input('Enter Book Price:'))
   brec = str(bno) + ',' + bname +','+author+','+str(price)+'\n'
   myfile.write(brec)
   ans=input('Add more….')
myfile.close()
```

Example: copy the content of one file to another file

```
myfile1 = open('book.txt','r')
myfile2 = open('book_copy.txt','w')
str1= " "
while str1:
   str1=myfile1.readline()
   myfile2.write(str1)
myfile1.close()
myfile2.close()
print('Files copied successfully….')
```

When we write any data to file, python hold everything in buffer (temporary memory) and pushes it onto actual file later. If you want to force Python to write the content of buffer onto storage, you can use flush() function.

```
myfile = open('temp.txt', 'w+')
myfile.write('Hello')
myfile.write('Welcome')
n = input('Press any key…')
myfile.write('How are you?')
myfile.close()
```

Python automatically flushes the files when closing them i.e. it will be implicitly called by the close(), BUT if you want to flush before closing any file you can use flush()

```
myfile = open('temp.txt', 'w+')
myfile.write('Hello')
myfile.write('Welcome')
myfile.flush()
n = input('Press any key...')
myfile.write('How are you?')
myfile.close()
```

Removing whitespaces after reading from file

read() and readline() reads data from file and return it in the form of string and readlines() returns data in the form of list.

All these read function also read leading and trailing whitespaces, new line characters. If you want to remove these characters you can use functions

strip() : removes the given character from both ends.
lstrip(): removes given character from left end
rstrip(): removes given character from right end

```
myfile = open('testfile.txt')
line1=myfile.readline()
print('length of line is:', len(line1))
line1=line1.rstrip('\n')
print('length of line is:', len(line1))
line2=myfile.readline()
print('length of line is:', len(line2))
line2=line2.lstrip()
print('length of line is:', len(line2))
```

File Pointer:

Every file maintains a file pointer which tells the current position in the file where reading and writing operation will take.

When we perform any read/write operation two things happens:

- The operation at the current position of file pointer
- File pointer advances by the specified number of bytes.

The tell () method of python tells us the current position within the file, whereas The seek(offset[, from]) method changes the current file position. If from is 0, the beginning of the file to seek. If it is set to 1, the current position is used. If it is set to 2 then the end of the file would be taken as seek position. The offset argument indicates the number of bytes to be moved.

```
f = open("a.txt", 'w')
line = 'Welcome to India \n Enjoy Indian Dishes'
f.write(line)
f.close()
```

```
f = open("a.txt", 'rb+')
print(f.tell())
print(f.read(7)) # read seven characters
print(f.tell())
print(f.read())
print(f.tell())
f.seek(9,0) # moves to 9 position from begining
print(f.read(5))
f.seek(4, 1) # moves to 4 position from current location
print(f.read(5))
f.seek(-5, 2) # Go to the 5th byte before the end
print(f.read(5))
f.close()
```

File Modes and Opening position of file pointer

FILE MODE	OPENING POSITION
r, r+, rb, rb+, r+b	Beginning of file
w, w+, wb, wb+, w+b	Beginning of file (overwrites the file if file already exists
a, ab, a+, ab+, a+b	At the end of file if file exists otherwise creates a new file

Standard Input/Output/Error

Standard Input : Keyboard
Standard Output: Monitor
Standard Error : Monitor

Standard Input devices(stdin) reads from keyboard
Standard output devices(stdout) display output on monitor
Standard error devices(stderr) same as stdout but normally for errors only.

Most programs make output to "standard out",input from "standard in", and error messages go to standard error).standard output is to monitor and standard input is from keyboard.

e.g.program

```
import sys
a = sys.stdin.readline()
sys.stdout.write(a)
a = sys.stdin.read(5)#entered 10 characters.a contains 5 characters.
                     #The remaining characters are waiting to be read.
sys.stdout.write(a)
b = sys.stdin.read(5)
sys.stdout.write(b)
sys.stderr.write("\ncustom error message")
```

Absolute Path & Relative Path:

Absolute path is the full address of any file or folder from the Drive i.e. from ROOT FOLDER. It is like:
Drive_Name:\Folder\Folder…\filename

Relative Path is the location of file/folder from the current folder. To use Relative path special symbols are:
Single Dot (.) : single dot (.) refers to current folder.
Double Dot (..) : double dot (..) refers to parent folder
Backslash (\ ) : first backslash before (.) and double dot(..)refers to ROOT folder.

Absolute path: C:/users/admin/docs/staff.txt
If PWD is C:/users/admin/, then the relative path to staff.txt would be: docs/staff.txt
Note, PWD + relative path = absolute path.
os.chdir("C:/users/admin/docs")
os.path.exists("staff.txt")
This returns TRUE if stuff.txt exists and it works.
Now, instead if we write, os.path.exists("C:/users/admin/docs/staff.txt")
This will returns TRUE.

If we don't know where the user executing the script from, it is best to compute the absolute path on the user's system using os and __fil__e.
__file__ is a global variable set on every Python script that returns the relative path to the *.py file that contains it.

e.g.program import os
print(os.getcwd())
os.mkdir("newdir1")
os.chdir("newdir1")
print(os.getcwd())
my_absolute_dirpath = os.path.abspath(os.path.dirname(__file__))
print(my_absolute_dirpath)

Binary File Handling

To store string in binary file, we must convert it to binary format either by prefixing the string with 'b' or using the encode() function.

Example: storing string into Binary file

```
str1='welcome'
f=open('myfile.bin','wb')
f.write(str1.encode())
f.write(b' to India')
f.close()
```

We can use 'a' in place of 'w' for append

Example: reading string from binary file

```
f=open('myfile.bin','rb')
str1=f.read()
print(str1)
print(str1.decode())
f.close()
```

We can observe, without decoding it will prefix text with 'b

```
# program to create binary file and storing data
rec_size =15 # each data will take 15 byte
with open('name.dat','wb') as f:
    ans='y'
    while ans.lower()=='y':
        name=input('Enter name:')
        length=len(name)
        name = name + (rec_size-length)* ' ' #to add extra space
        name = name.encode()
        f.write(name)
        ans=input('Do you want to add more?')
```

```
# program to randomly access data
# first data will be stored at 0 and second will on 15..
```

```
rec_size = 15
num = int(input('Enter record number :'))
with open('name.dat', 'rb') as f:
# f=open('name.dat', 'rb')
   f.seek(rec_size*(num-1)) #setting read pointer at desired location
   str=f.read(rec_size)
   if(len(str)==0):
      print('Incorrect position !!')
   else:
      print(str.decode())
f.close()
```

```
# program to search any data from file and display record number
import os
rec_size = 15
#finding the size of file
size=os.path.getsize('name.dat')
print('Size of file is:', size)
#finding the number of records
num_rec = int(size/rec_size)
print('Number of records: ', num_rec)
with open('name.dat','rb') as f: # f= open('name.dat','rb')
   n=input('Enter name to search:')
   n=n.encode()
   position=0
   found=False
   for i in range(num_rec):
      f.seek(position)
      str=f.read(15)
      if n in str:
         print('Found at record#',(i+1))
         found = True
      position+=rec_size
if not found:
      print('name not found!!')
```

```
# program to update names(data) in Binary file
import os
rec_size = 15
#finding the size of file
size=os.path.getsize('name.dat')
print('Size of file is:', size)
#finding the number of records
num_rec = int(size/rec_size)
print('Number of records: ', num_rec)

with open('name.dat','r+b') as f:
   old_name=input('Enter Name:')
   old_name=old_name.encode()
   new_name=input('Enter New Name:')
   ln=len(new_name)
   new_name=new_name+(15-ln)* ''
   new_name=new_name.encode()
   position=0
   found=False
   for i in range(num_rec):
      f.seek(position)
      str=f.read(15) #read each name
      if old_name in str:
         print('Updated Record Number',(i+1))
         found=True
         f.seek(-15,1) #sending cursor 15 byte back
         f.write(new_name)
      position+=rec_size
if not found:
   print('Name not found!!')
```

```
#program to delete data in binary file
import os
rec_size = 15
# finding size of file
size=os.path.getsize('name.dat')
print('Size of File :', size)
# finding Number of records

num_rec = int(size/rec_size)
print('Number of records:', num_rec)

f1=open('name.dat','rb')
f2=open('newname.dat','wb')
nm=input('Enter name to be deleted: ')
l=len(nm)
nm=nm+(rec_size-l)* ' '
nm=nm.encode()

position=0
found=False
for i in range(num_rec):
   str=f1.read(rec_size)
   if(str!=nm):
      f2.write(str)
print('Record deleted!!')
f1.close()
f2.close()
os.remove('name.dat')
os.rename('newname.dat','name.dat')
```

Example:

```
binary_file=open("D:\\binary_file.dat",mode="wb+")
text="Hello 123"
encoded=text.encode("utf-8")
binary_file.write(encoded)
binary_file.seek(0)
binary_data=binary_file.read()
print("binary:",binary_data)
text=binary_data.decode("utf-8")
print("Decoded data:",text)
```

In above program binary_file.dat is opened in wb+ mode so that after writing, reading operation can be done on binary file. String variable text hold text to be encoded before writing with the help of encode method(). utf-8 is encoding scheme. After writing text, we again set reading pointer at beginning with the help of seek() method. then read the text from file and decode it with the help of decode() method then display the text.

It is not very easy to use when we want to write several objects into the binary file i.e. string, integer, a list or dictionary into the file. How would we read the contents of this file later? Python has a module which does this work for us and is extremely easy to use. This module is called **pickle**; it provides us with the ability to serialize and deserialize objects, i.e., to convert objects into bit streams which can be stored into files and later be used to reconstruct the original objects.

pickle.dump() function is used to store the object data to the file.
It takes 3 arguments.

1. First argument is the object that we want to store.
2. The second argument is the file object we get by opening the desired file in write-binary (wb) mode.
3. Third argument is the key-value argument.

Pickle.load() function is used to retrieve pickled data. The steps are quite simple. We have to use pickle.load() function to do that.
The primary argument of pickle load function is the file object that you get by opening the file in read-binary (rb) mode.

Example (Binary file R/W Operation)

```
import pickle
output_file = open("d:\\a.bin", "wb")
myint = 42
mystring = "Sample Text"
mylist = ["python", "sql", "mysql"]
mydict = { "name": "ABC", "job": "XYZ" }
pickle.dump(myint, output_file)
pickle.dump(mystring, output_file)
pickle.dump(mylist, output_file)
pickle.dump(mydict, output_file)
output_file.close()
input_file = open("d:\\a.bin", "rb")
myint = pickle.load(input_file)
mystring = pickle.load(input_file)
mylist = pickle.load(input_file)
mydict = pickle.load(input_file)
print("myint = " , myint)
print("mystring = " , mystring)
print("mylist = " , mylist)
print("mydict = " , mydict)
input_file.close()
```

Example (Binary file R/W Operation iteration over)

```
import pickle
output_file = open("d:\\a.bin", "wb")
myint = 42
mystring = "Sample Text"
mylist = ["python", "sql", "mysql"]
mydict = { "name": "ABC", "job": "XYZ" }
pickle.dump(myint, output_file)
pickle.dump(mystring, output_file)
pickle.dump(mylist, output_file)
pickle.dump(mydict, output_file)
output_file.close()
with open("d:\\a.bin", "rb") as f:
   while True:
      try:
         r=pickle.load(f)
         print(r)
         print('Next data')
      except EOFError:  break
f.close()
```

Example (Insert/append record in a Binary file)

```
import pickle
rollno = int(input('Enter roll number:'))
name = input('Enter Name:')
marks = int(input('Enter Marks'))

#Creating the dictionary
rec = {'Rollno':rollno,'Name':name,'Marks':marks}
#Writing the Dictionary
f = open('d:/student.dat','ab')
pickle.dump(rec,f)
f.close()
```

Example (Read records from a Binary file)

```
import pickle
f = open('d:/student.dat','rb')
while True:
    try:
        rec = pickle.load(f)
        print('Roll Num:',rec['Rollno'])
        print('Name:',rec['Name'])
        print('Marks:',rec['Marks'])
    except EOFError:
        break
f.close()
```

Example (Search record in a Binary file)

```
import pickle
f = open('d:/student.dat','rb')
flag = False
r=int(input('Enter rollno to be searched'))
while True:
    try:
        rec = pickle.load(f)
        if rec['Rollno'] == r:
            print('Roll Num:',rec['Rollno'])
            print('Name:',rec['Name'])
            print('Marks:',rec['Marks'])
            flag = True
    except EOFError:
        break
if flag == False:
    print('No Records found')
```

```
f.close()
```

Example (Update record of a Binary file)

```
import pickle
f = open('d:/student.dat','rb')
reclst = []
r=int(input('enter roll no to be updated'))
m=int(input('enter correct marks'))
while True:
   try:
rec = pickle.load(f)
      reclst.append(rec)
   except EOFError:
      break
f.close()
for i in range (len(reclst)):
   if reclst[i]['Rollno']==r:
      reclst[i]['Marks'] = m
f = open('d:/student.dat','wb')
for x in reclst:
   pickle.dump(x,f)
f.close()
```

Example (Delete record of a Binary file)

```
import pickle
f = open('d:/student.dat','rb')
reclst = []
r=int(input('enter roll no to be deleted'))
while True:
   try:
      rec = pickle.load(f)
      reclst.append(rec)
   except EOFError:
      break
f.close()
f = open('d:/student.dat','wb')
for x in reclst:
   if x['Rollno']==r:
      continue
   pickle.dump(x,f)
f.close()
```

Example: Storing employee details in binary file

```
import pickle
emp=[]
f=open('employee.dat','wb')
ans='y'
while ans=='y':
   eno=int(input('Enter employee Number: '))
   name=input('Enter Employee name: ')
   salary=int(input('Enter Employee Salary: '))
   emp.append([eno,name,salary])
   ans=input('Add More records?')
pickle.dump(emp,f)
f.close()
```

Example: Reading and Display Record

```
import pickle
emp =[]
f = open('employee.dat','rb')
ans ='y'
while True:
   try:
      emp = pickle.load(f) #loading data in emp list
   except EOFError:
      break
for e in emp:
   print(e)
f.close()
```

Example: Display Record (Formatted Output)

```
import pickle
emp = []
f = open('employee.dat', 'rb')
ans ='y'
while True:
   try:
      emp = pickle.load(f) #loading data in emp list
   except EOFError:
      break
print("%10s"%"Emp No","%20s"%"Emp name","%10s"%"Emp salary")
for e in emp:
   print("%10s"%e[0],"%20s"%e[1],"%10s"%e[2])
```

```
f.close()
```

Example: Searching in Binary File

```
import pickle
emp=[]
f=open('employee.dat','rb')
ans='y'
print('search employee')
en=int(input('enter employee number'))
found=False
while True:
   try:
      emp=pickle.load(f)
   except EOFError:
      break
print("%10s"%"Emp No","%20s"%"Emp Name","%10s"%"Emp Salary")
for e in emp:
   if(e[0]==en):
      print("%10s"%e[0],"%20s"%e[1],"%10s"%e[2])
      found=True
      break
if found==False:
   print('Not found')
f.close()
```

Example: Finding Number of Record in Binary File

```
import pickle
import os
emp=[]
f=open('employee.dat','rb')
emp=pickle.load(f)
l=len(emp) #count the number of objects
while True:
   try:
      emp=pickle.load(f)
   except EOFError:
      break
print("%10s"%"Emp No","%20s"%"Emp Name","%10s"%"Emp Salary")
for e in emp:
   print("%10s"%e[0],"%20s"%e[1],"%10s"%e[2])
print('Total records are:',l)
f.close()
```

Example: Updating Employee Record

```
import pickle
emp=[]
f=open('employee.dat','rb')
emp=pickle.load(f)
print('record of employee')
print(emp)
f.close()
f=open('employee.dat','wb')
found=False
en=int(input('enter employee number to update'))
for i in range(len(emp)):
   if emp[i][0]==en:
      sal=int(input('Enter new salary'))
      emp[i][2]=sal
      found=True
      print('Record updated')
if found==False:
   print('Not record found')
pickle.dump(emp,f)
f.close()
f=open('employee.dat','rb')
emp=pickle.load(f)
print('record of employee after updation')
print(emp)
f.close()
```

Example: Deleting Employee Record

```
import pickle
emp=[]
f=open('employee.dat','rb')
emp=pickle.load(f)
print('record of employee')
print(emp)
f.close()
f=open('employee.dat','wb')
en=int(input('enter employee number to delete'))
emp1=[]
for i in range(len(emp)):
   if emp[i][0]!=en:
      emp1.append(emp[i])
pickle.dump(emp1,f)
f.close()

f=open('employee.dat','rb')
emp=[]
emp=pickle.load(f)
print('record of employee after deletion')
print(emp)
f.close()
```

CSV File Handling

CSV is a simple file format used to store tabular data, such as a spread sheet or database. CSV stands for "comma-separated values".

A comma-separated values file is a delimited text file that uses a comma to separate values.

Each line of the file is a data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format.

Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel or Open Office Calc.

To perform read and write operation with CSV file, we need to import csv module.

open() function is used to open file, and return file object.

Steps to working on CSV files:

- import csv module
- Use open() to open csv file, it will return file object.
- Pass this file object to reader object.
- Perform operation as desired

How to create CSV file

Using MS Excel (after preparing a data file save it as csv)
Using notepad (typing record separated by , and one record in each line)
Save as "filename.csv"

Reading from csv file

```
import csv
with open('csvt.csv') as csvfile:
   myreader = csv.reader(csvfile,delimiter=',')
   print("%10s"%"empno","%20s"%"emp name","%10s"%"salary")
   print("      ==========================")
   for row in myreader:
      print("%10s"%row[0],"%10s"%row[1],"%10s"%row[2])
```

Counting number of records

```
import csv
with open('csvt.csv') as csvfile:
   myreader = csv.reader(csvfile,delimiter=',')
   count=0
   print("%10s"%"empno","%20s"%"emp name","%10s"%"salary")
   print("==============================")
   for row in myreader:
      print("%10s"%row[0],"%20s"%row[1],"%10s"%row[2])
      count+=1
   print("==============================")
   print("%30s"%"total record:",count)
print("==============================")
```

Sum of Salary and counting employee getting more than 7000

```
import csv
with open('csvt.csv') as csvfile:
   myreader = csv.reader(csvfile,delimiter=',')
   count =0
   sum =0
   print("%10s"%"empno","%20s"%"emp name","%10s"%"salary")
   print('================================')
   for row in myreader:
      print("%10s"%row[0],"%20s"%row[1],"%10s"%row[2])
      sum+=int(row[2])
      if int (row[2])>70000:
         count+=1
   print('================================')
    print('%30s'%'sum of salary',sum)
   print('%40s'%'# emp getting more than 70000 :',count)
   print('================================')
```

Writing data in CSV file

- import csv module
- Use open() to open CSV file by specifying mode "w" or "a", it will return file object.
- "w" will overwrite previous content
- "a" will add content to the end of previous content.
- Pass the file object to writer object with delimiter.
- Then use writerow() to send data in CSV file

```
import csv
with open('csvt.csv', mode='a') as csvfile:
    mywriter = csv.writer(csvfile,delimiter=',')
    ans='y'
    while ans.lower() =='y':
        eno=int(input('Enter Emp number:'))
        name=input('Enter Emp name:')
        salary=int(input('Enter Emp Salary:'))
        mywriter.writerow([eno,name,salary])
        print('data saved…')
        ans=input('Add More….')
```