Chapter-File Handling

Previous Knowledge

- How to read and write from console I/O?

- Reading:

raw_input and input

- Writing:

Print statement

- Difference between raw_input and input?
- *The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.*

By: Vijeta Vashisth

Learning Outcomes

- **File handling:** Need for a data file
- **Types of file:** Text files, Binary files and CSV files.
- **Text files:** Basic operations on a text file: Open (filename – absolute or relative path, mode) / Close a text file, Reading and Manipulation of data from a text file, Appending data into a text file, standard input / output and error streams, relative and absolute paths.
- **Binary files:** Basic operations on a binary file: Open (filename – absolute or relative path, mode) / Close a binary file, Pickle Module- methods load and dump; Read, Write/Create, Search, Append and update operations.
- **CSV files:** Import csv module, functions – Open / Close a csv file, Read from a csv file and Write into a csv file using csv.reader ( ) and csv.writerow( ).

By: Vijeta Vashisth

Introduction

- To Store data in organized manner
- To store data permanently
- To access data faster
- To Search data faster
- To easily modify data later on

- 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.
- In Python, File Handling consists of following three steps:
  Open the file.
  Process file i.e. perform read or write operation.
  Close the file.

By: Vijeta Vashisth

Types of File

- There are two types of files:
  Text Files- A file 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 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.

By: Vijeta Vashisth

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.

OPENING AND CLOSING A FILE

- **Before any reading or writing operation of any file, it must be opened first.**
- Python provide built in function open() for it. On calling of this function creates file object for file operations.

  Syntax

  file object = open(<file_name>, <access_mode>,< buffering>)

  file_name :- name of the file ,enclosed in double quotes.

  access_mode:- Determines the what kind of operations can be performed with file,like read,write etc.

  Buffering:- 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 and if it is negative then buffer size is system default.

By: Vijeta Vashisth

File Modes

Sr. No.	Mode & Description
1	**r** - reading only.Sets file pointer at beginning of the file . This is the default mode.
2	**rb** – same as r mode but with binary file
3	**r+** - both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** - same as r+ mode but with binary file
5	**w** - writing only. Overwrites the file if the file exists. If not, creates a new file for writing.
6	**wb** – same as w mode but with binary file.
7	**w+** - both writing and reading. Overwrites . If no file exist, creates a new file for R & W.
8	**wb+** - same as w+ mode but with binary file.
9	**a** -for appending. Move file pointer at end of the file.Creates new file for writing,if not exist.
10	**ab** – same as a but with binary file.
11	**a+** - for both appending and reading. Move file pointer at end. If the file does not exist, it creates a new file for reading and writing.
12	**ab+** - same as a+ mode but with binary mode.

By: Vijeta Vashisth

Basic Text file operations

- Open (filename – absolute or relative path, mode)
- Close a text file
- Reading/Writing data
- Manipulation of data
- Appending data into a text file

By: Vijeta Vashisth

METHODS OF OS MODULE

- Before Starting file operation following methods are used by programmer to perform file system related methods and are available in os module(standard module) which can be used during file operations.
- **Rename() method:** used to rename the file.
  Syntax: os.rename(current_file_name, new_file_name)
- **remove() method:** to delete file.
  Syntax: os.remove(file_name)
- **mkdir() method:** to create directories in the current directory.
  Syntax: os.mkdir("newdir")
- ***chdir()* method:** to change the current directory.
  Syntax: os.chdir("newdir")
- **getcwd() method:** displays the current directory.
  Syntax: os.getcwd()
- **rmdir() method:** deletes the directory.
  Syntax: os.rmdir('dirname')

By: Vijeta Vashisth

Absolute Path vs Relative Path

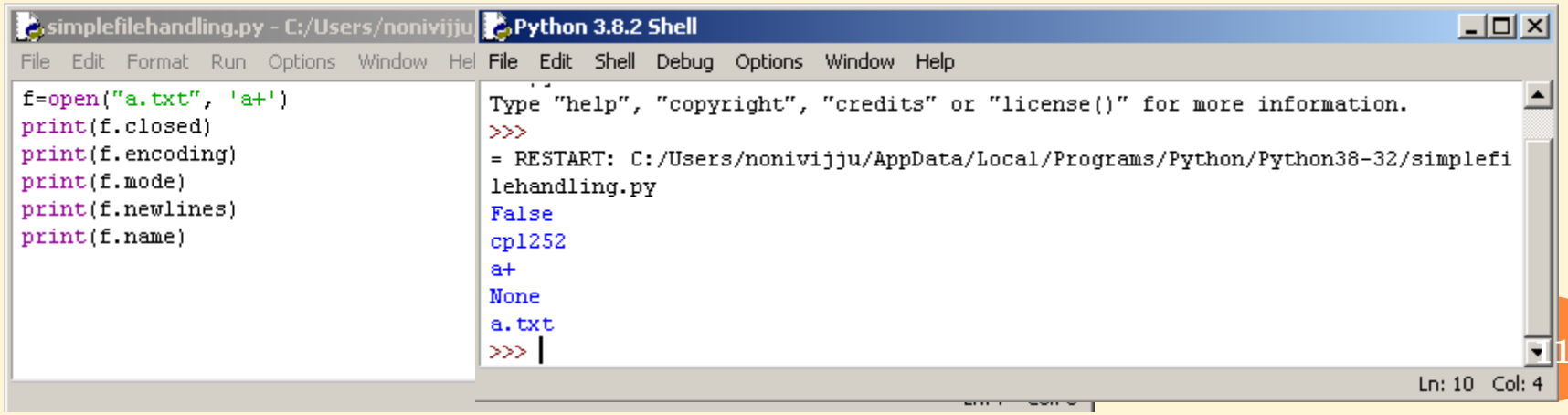
- The absolute path is the full path to some place on your computer. The relative path is the path to some file with respect to your current working directory (PWD).
  Example:

- 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 __file__. __file__ is a global variable set on every Python script that returns the relative path to the *.py file that contains it.

By: Vijeta Vashisth

File object attributes / 1. open a text file

- closed: It returns true if the file is closed and false when the file is open.
- encoding: Encoding used for byte string conversion.
- mode: Returns file opening mode
- name: Returns the name of the file which file object holds.
- newlines: Returns “\r”, “\n”, “\r\n”, None or a tuple containing all the newline types seen.



2. Close a text file

- **close():** used to close an open file. After using this method,an opened file will be closed and a closed file cannot be read or written any more.

- **E.g. program**

```
f = open("a.txt", 'a+')
print(f.closed)
print("Name of the file is",f.name)
f.close()
print(f.closed)
```

OUTPUT:

False

Name of the file is a.txt

true

By: Vijeta Vashisth

READ/WRITE TEXT FILE

- **The write() Method**
  It writes the contents to the file in the form of string. It does not return value. Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.
- **The read() Method**
  It reads the entire file and returns it contents in the form of a string. Reads at most size bytes or less if end of file occurs. If size not mentioned then read the entire file contents.
- **Example:**

```
f = open("a.txt", 'w')
line1 = ‘my first file handling program’
f.write(line1)
line2="\n I love python programming"
f.write(line2)
f.close()
f = open("a.txt", 'r')
text = f.read()
print(text)
f.close()
```

READ TEXT FILE

- **readline([size]) method:** Read no of characters from file if size is mentioned till EOF or till new line character.
- e.g. program:
- f = open("a.txt", 'w')
  line1 = 'my first file handling program'
  f.write(line1)
  line2="\n I love python programming"
  f.write(line2)
  f.close()
  f = open("a.txt", 'r')
  text = f.readline()
  print(text)
  text= f.readline()
  print(text)
  f.close()

By: Vijeta Vashisth

READ TEXT FILE

- **readlines([size]) method:** Read no of lines from file if size is mentioned or read all the content if size is not mentioned.
- **program:**

```
f = open("a.txt", 'w')
line1 = ‘my first file handling program’
f.write(line1)
line2="\n I love python programming"
f.write(line2)
f.close()
f = open("a.txt", 'r')
text = f.readlines(1)
print(text)
    f.close()
```

OUTPUT: my first file handling program

By: Vijeta Vashisth

READ TEXT FILE

- **Iterating over lines in a file:**
- **Program:**

```
f = open("a.txt", 'w')
line1 = ‘my first file handling program’
f.write(line1)
line2="\n I love python programming"
f.write(line2)
f.close()
f = open("a.txt", 'r')
for text in f.readlines():
print(text)
    f.close()
```

By: Vijeta Vashisth

READ TEXT FILE

- **Program:**

```
f = open("a.txt", 'w')
line1 = ‘my first file handling program’
f.write(line1)
line2="\n I love python programming"
f.write(line2)
f.close()
f = open("a.txt", 'r')
for text in f.readlines():
        for word in text.split():
            print(word)
    f.close()
```

By: Vijeta Vashisth

PRACTICE TIME:

- **Write a program to count the number of characters from a file. (Don't count white spaces)**
- **WAP to count the number of words in a file**
- **WAP to count number of vowels in a text file**

By: Vijeta Vashisth

File positions

- There are two methods which points the file pointer:
- **Tell() method**
- **Seek() method**
- The *tell()* method of python tells us the current position within the file, where as 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.

```
# creating a file
f = open('file1.txt', 'w')
 f.write('This is line1.\n')
f.write('This is line2.\n')
f.write('This is line3.\n')
f.close()

# now, reading operations ....opening the file
f = open('file1.txt', 'r')
# reading  first10 characters
str = f.read(10);
print('str: ', str)

# Check current offset/position
offset = f.tell();
print('Current file offset: ', offset)

# Reposition pointer at the beginning once again
offset = f.seek(0, 0);
str = f.read(10);
print('Again the str: ', str)
f.close()
```

By: Vijeta Vashisth

Modify a Text file

- We can append to a file or overwrite part of it using seek() function, but if we want to add text at the beginning or the middle, we'll have to rewrite it.
- It is an operating system task, not a Python task. It is the same in all languages.
- Steps involve in modification:
- Read the file.
- Make changes at the desired position.
- Write it out into a new file e.g. temp.txt.
- Renaming it as original file.

 # note: This is a good, safe way to do it because if the file write crashes or aborts for any reason in between, we still have our untouched original file.

By: Vijeta Vashisth

Example-1 Replace string in the same File

- fin = open("dummy.txt", "rt")
  data = fin.read()
  data = data.replace('my', 'your')
  fin.close()
  fin = open("dummy.txt", "wt")
  fin.write(data)
  fin.close()
- Open file dummy.txt in read text mode rt.
- fin.read() reads whole text in dummy.txt to the variable data.
- data.replace() replaces all the occurrences of my with your in the whole text.
- Finally, fin.close() closes the input file dummy.txt

By: Vijeta Vashisth

Example-2 Replace string using temporary file

```
import os
f=open("d:\\a.txt","r")
g=open("d:\\c.txt","w")
for text in f.readlines():
text=text.replace('my','your')
g.write(text)
f.close()
g.close()
os.remove("d:\\a.txt")
os.rename("d:\\c.txt","d:\\a.txt")
print("file contents modified")
```

By: Vijeta Vashisth

Appending content to a Text file

- Steps:
- First open the file in append mode(if exists).
- Then write data into the file.
- Then open the file in read mode and read the data.

By: Vijeta Vashisth

Write a program to take the details of book from the user and write the record in text file.

```
fout=open("D:\\python programs\\Book.txt",'w')
n=int(input("How many records you want to write in a file ? :"))
for i in range(n):
    print("Enter details of record :", i+1)
    title=input("Enter the title of book : ")
    price=float(input("Enter price of the book: "))
    record=title+" , "+str(price)+'\n'
    fout.write(record)
    fout.close( )
```

By: Vijeta Vashisth

Write a program to take the details of book from the user and write the record in the end of the text file.

```
fout=open("D:\\python programs\\Book.txt",'a')
n=int(input("How many records you want to
write in a file ? :"))
for i in range(n):
        print("Enter details of record :", i+1)
        title=input("Enter the title of book : ")
        price=float(input("Enter price of the book: "))
        record=title+" , "+str(price)+'\n'
        fout.write(record)
        fout.close( )
```

By: Vijeta Vashisth

STANDARD INPUT, OUTPUT, AND ERROR STREAMS IN PYTHON

- Standard output is provided on the monitor and standard input is from keyboard.
- Most programs make output to "standard out“, input from "standard in“, and error messages go to standard error).
- Example:
- 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")

By: Vijeta Vashisth

Delete a file

- To delete a file, you have to import the **os** module, and use **remove( )** function.

- import os
  os.remove("Book.txt")

By: Vijeta Vashisth

Check if File exist

- To avoid getting an error, you might want to check if the file exists before you try to delete it

- Example:
- import os
  if os.path.exists("Book.txt"):
  os.remove("Book.txt")
  else:
  print("The file does not exist")

By: Vijeta Vashisth

Binary File

- Binary files store data in the binary format (0's and 1's) which is understandable by the machine. So when we open the binary file in our machine, it decodes the data and displays in a human-readable format.

By: Vijeta Vashisth

WRITING TO A BINARY FILE

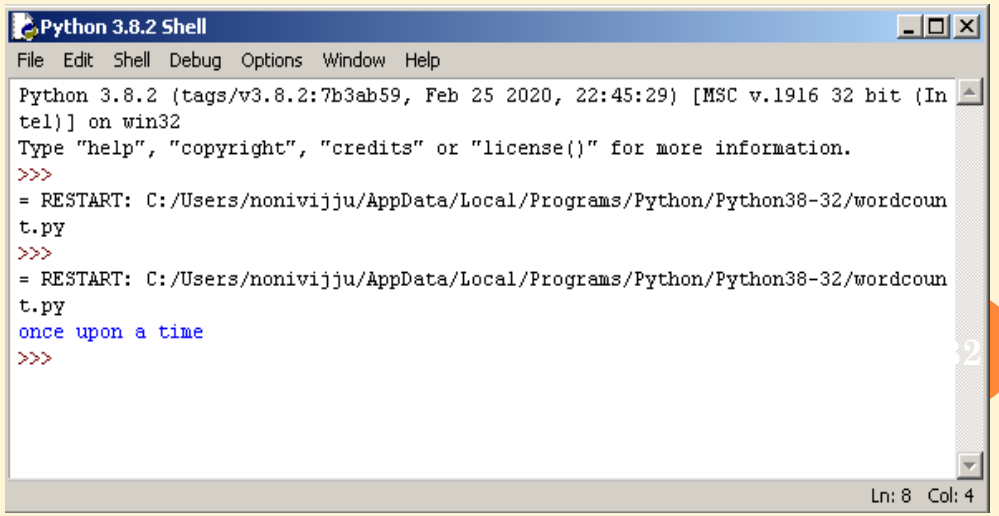
- Example:

```
#opening a file in read and write mode
 fout= open("story.dat", "wb")
 normal_str= "once upon a time"
#encoding normal text
 encoded_str= normal_str.encode("ASCII")
#write data into file
 fout.write(encoded_str)
 fout.close()
```

READING FROM A BINARY FILE

- Example:

```
#open a binary file in read and binary mode
    fin=open("story.dat", "rb")
    str1= fin.read()
 # reconverting into normal text
    decoded_str= str1.decode("ASCII")
# printing the data
    print(decoded_str)
    fin.close()
```



Pickle Module

- Reading and writing in the previous format is tedious when working with several objects.
- Example: when we want to write data with variable length such as list, dictionary and string in a single program. Thus, encoding and decoding becomes difficult.
- To overcome this problem, python provides the **pickle module** which provides the ability to serialize and deserialize objects
- It convert objects into bitstreams which can be stored into files and later on used to reconstruct the original objects.

By: Vijeta Vashisth

PICKLE DUMP FUNCTION

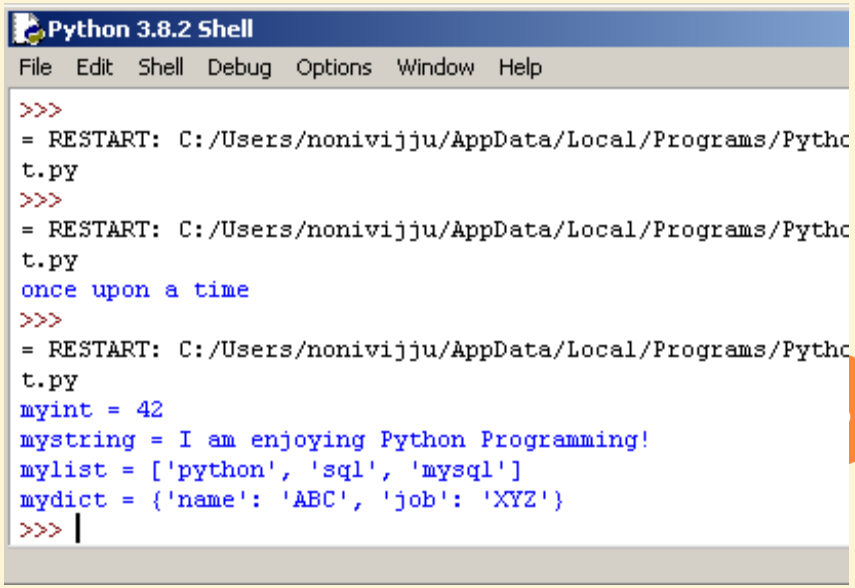
- **pickle.dump()** function is used to store the object data to the file. It takes 3 arguments.
- First argument is the object that we want to store.
- The second argument is the file object we get by opening the desired file in write-binary (wb) mode.
- The third argument is the key-value argument. This argument defines the protocol. There are two type of protocol –

pickle.HIGHEST_PROTOCOL
pickle.DEFAULT_PROTOCOL.

By: Vijeta Vashisth

PICKLE LOAD FUNCTION

- **Pickle.load()** function is used to retrieve pickled data. The primary argument of pickle load function is the file object that we get by opening the file in read-binary (rb) mode.
- **Example:**
- import pickle
output_file = open("d:\\a.bin", "wb")
myint = 42
mystring = “I am enjoying Python Programming!"
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 = %s" % myint)
print("mystring = %s" % mystring)
print("mylist = %s" % mylist)
print("mydict = %s" % mydict)
input_file.close()



By: Vijeta Vashisth

Iteration over Binary file - pickle module

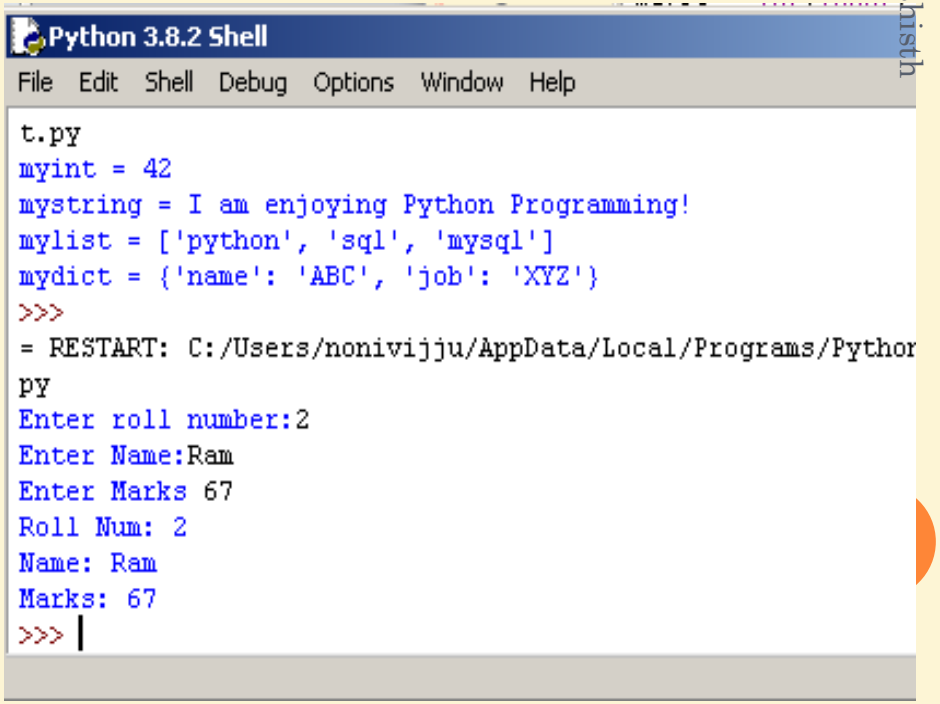
```
import pickle
output_file = open("d:\\a.bin", "wb")
myint = 42
mystring = “I am enjoying Python Programming!"
mylist = ["python", "sql", "mysql"]
mydict = { "name": "ABC", "job": "XYZ" }
pickle.dump(myint, output_file)
pickle.dump(mystring, output_file)
pickle.dump(mylist, output_file)
pickle.dump(mydict, 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()
```

Reading objects one by one

By: Vijeta Vashisth

Insert/append record in a Binary file

```
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()
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()
```



SEARCH RECORD IN A BINARY FILE

```
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()
```

UPDATE RECORD OF A BINARY FILE

```
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()
```

By: Vijeta Vashisth

Delete record of a Binary file

```
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()
```

By: Vijeta Vashisth

CSV FILES

- **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.
- **CSV File Characteristics**
  - One line for each record
  - Comma separated fields
  - Space-characters adjacent to commas are ignored
  - Fields with in-built commas are separated by double quote characters.
- **When Use CSV?**
  - When data has a strict tabular structure
  - To transfer large database between programs
  - To import and export data to office applications
  - To store, manage and modify shopping cart catalogue

By: Vijeta Vashisth

CSV FILES

- **CSV Advantages**
  - CSV is faster to handle
  - CSV is smaller in size
  - CSV is easy to generate
  - CSV is human readable and easy to edit manually
  - CSV is simple to implement and parse
  - CSV is processed by almost all existing applications
- **CSV Disadvantages**
  - No standard way to represent binary data
  - There is no distinction between text and numeric values
  - Poor support of special characters and control characters
  - CSV allows to move most basic data only. Complex configurations cannot be imported and exported this way
  - Problems with importing CSV into SQL (no distinction between NULL and quotes)

By: Vijeta Vashisth

WRITE / READ CSV FILE

```
import csv
#csv file writing code
with open('d:\\a.csv','w') as newFile:
newFileWriter = csv.writer(newFile)
newFileWriter.writerow(['user_id','beneficiary'])
newFileWriter.writerow([1,'xyz'])
newFileWriter.writerow([2,'pqr'])
newFile.close()
#csv file reading code
with open('d:\\a.csv','r') as newFile:
newFileReader = csv.reader(newFile)
for row in newFileReader:
print (row)
newFile.close()
```

By: Vijeta Vashisth

Question Session

By: Vijeta Vashisth