Ex. No: 9

Aim: *Read the following file formats*

a. Pickle files
b. Image files using PIL
c. Multiple files using Glob
d. Importing data from database

9 (A):

Pickle files:

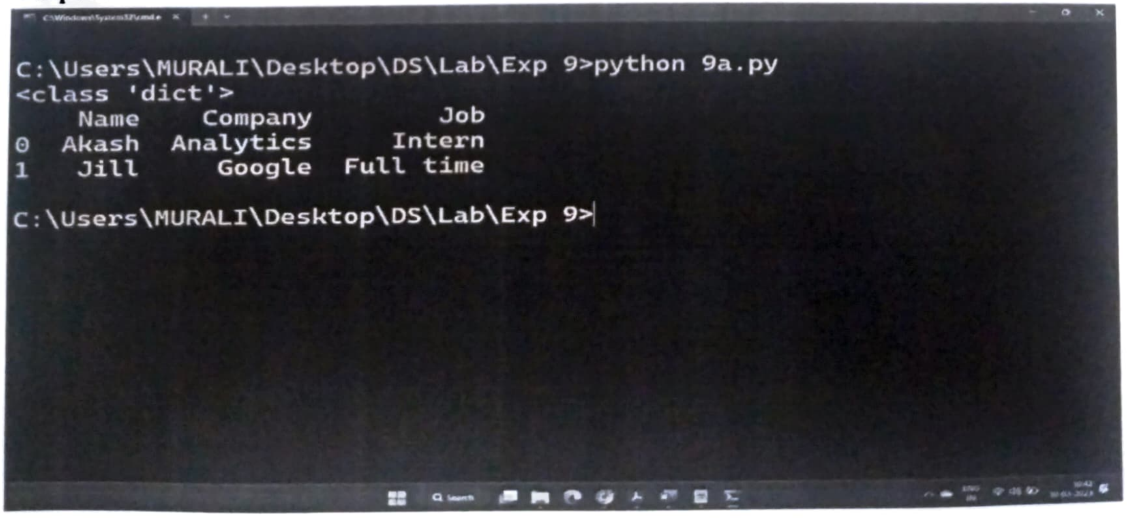
- Pickle files are used to store the serialized form of Python objects. This means objects like list, set, tuple, dict, etc. are converted to a character stream before being stored on the disk.
- This allows us to continue working with the objects later on. These are particularly useful when we have trained our machine learning model and want to save them to make predictions later on.
- So, if you serialized the files before saving them, you need to de-serialize them before you use them in your Python programs.
- This is done using the pickle.load() function in the pickle module. But when we open the pickle file with Python's open() function, we need to provide the 'rb' parameter to read the binary file.

Example:

9a.py

```
import pickle
import pandas as pd
emp_details = {"Name": {"0": "Akash", "1": "Jill"}, "Company":{"0": "Analytics", "1": "Google"}, "Job":
{"0": "Intern", "1": "Full time"}}
with open('Sample_pickle.pkl', 'wb') as f:
   pickle.dump(emp_details, f)
f.close()
with open('Sample_pickle.pkl','rb') as file:
   data = pickle.load(file)
# pickle data
print(type(data))
df_pkl = pd.DataFrame(data)
print(df_pkl)
```

Output:



```
C:\Users\MURALI\Desktop\DS\Lab\Exp 9>python 9a.py
<class 'dict'>
    Name    Company        Job
0  Akash  Analytics     Intern
1   Jill     Google  Full time

C:\Users\MURALI\Desktop\DS\Lab\Exp 9>
```

9 (B):

Image files using PIL:

- PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities.
- The Image module provides a class with the same name which is used to represent a PIL image.
- The module also provides a number of factory functions, including functions to load images from files, and to create new images.

Steps to Read an Image using PIL:

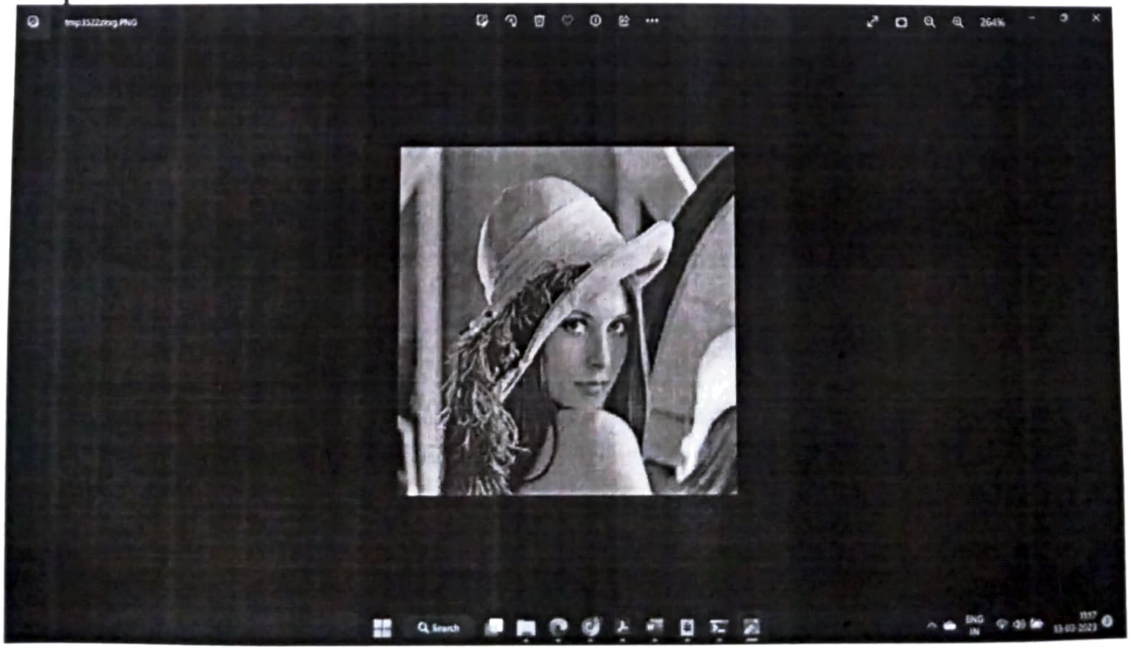
To read an image with Python Pillow library, follow these steps.

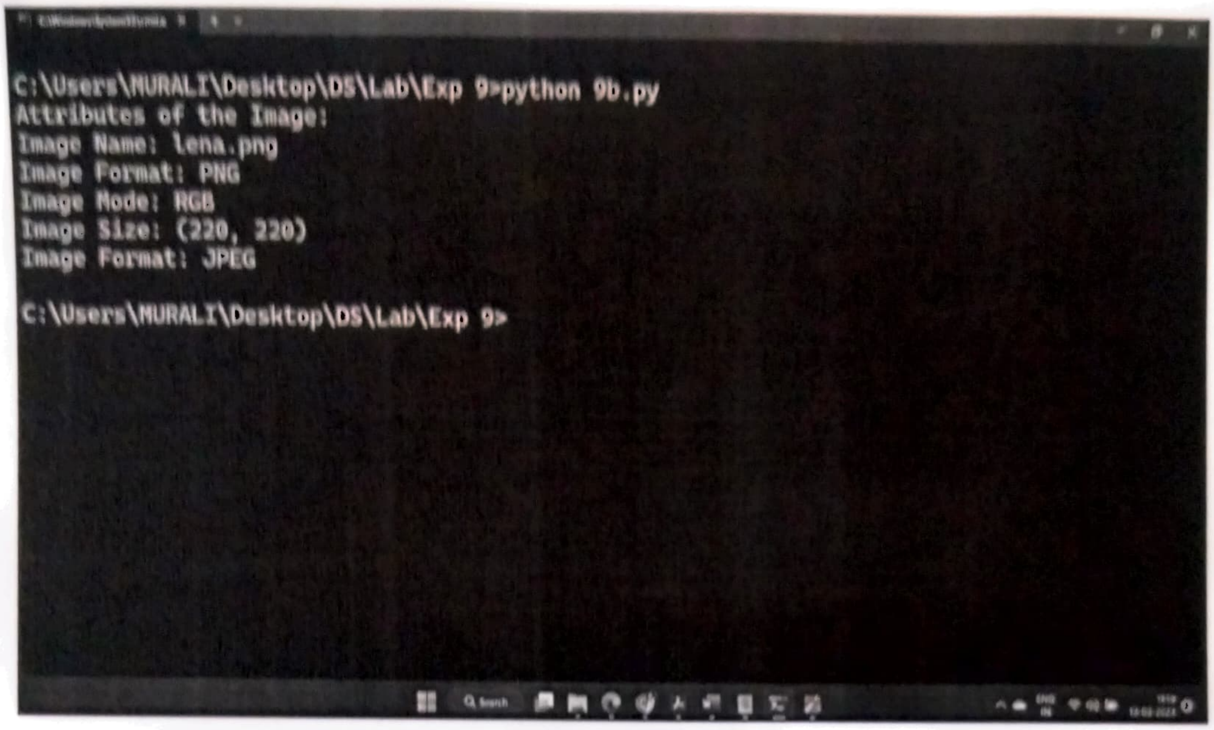
- Import Image from PIL library.
- Use Image.open() method and pass the path to image file as argument. Image.open() returns an Image object. You can store this image object and apply image operations on it.

Example:

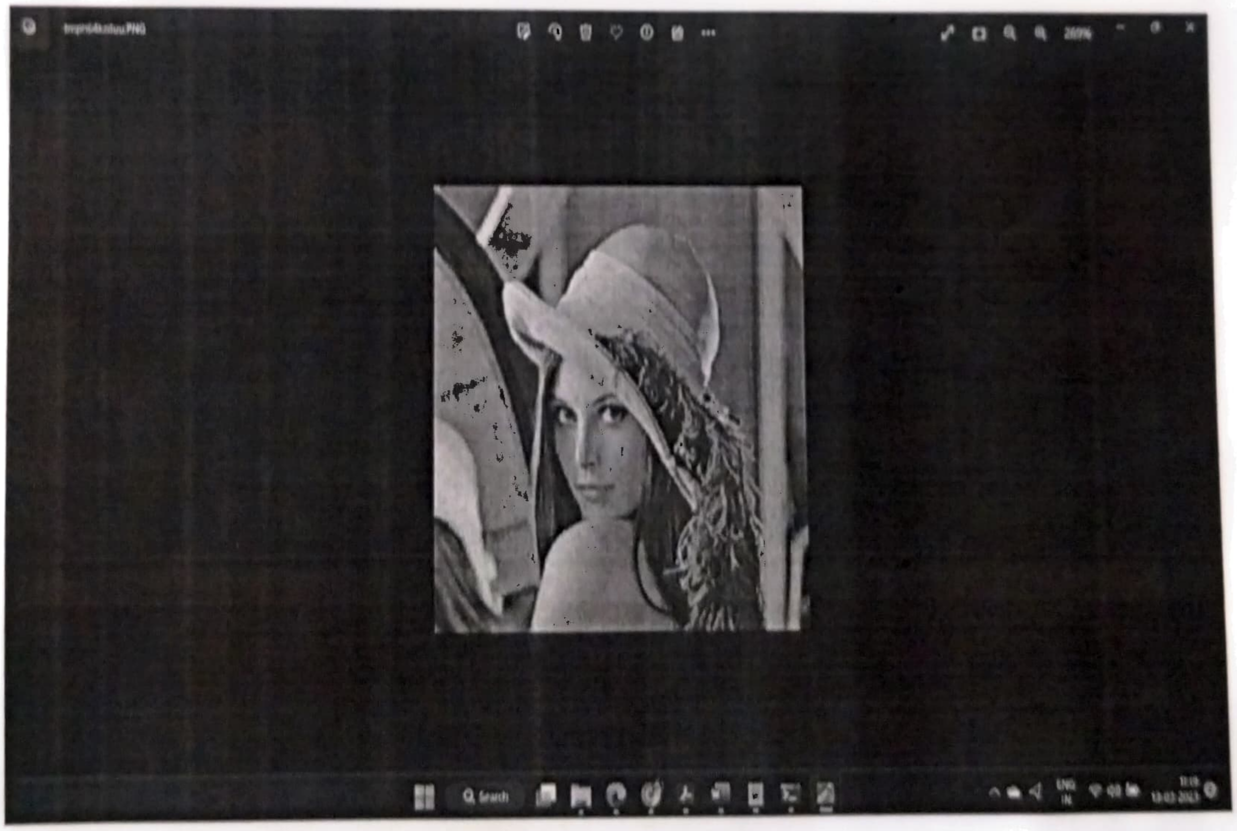
```
from PIL import Image
im = Image.open("lena.png")
im.show()
print("Attributes of the Image:")
print("Image Name:",im.filename)
print("Image Format:",im.format)
print("Image Mode:",im.mode)
print("Image Size:",im.size)
im.save("lena.jpeg")
im2 = Image.open("lena.jpeg")
print("Image Format:",im2.format)
fim=im2.transpose(Image.FLIP_LEFT_RIGHT)
fim.show()
```

Output:





```
C:\Users\MURALI\Desktop\DS\Lab\Exp 9>python 9b.py
Attributes of the Image:
Image Name: lena.png
Image Format: PNG
Image Mode: RGB
Image Size: (220, 220)
Image Format: JPEG

C:\Users\MURALI\Desktop\DS\Lab\Exp 9>
```



9 (C):

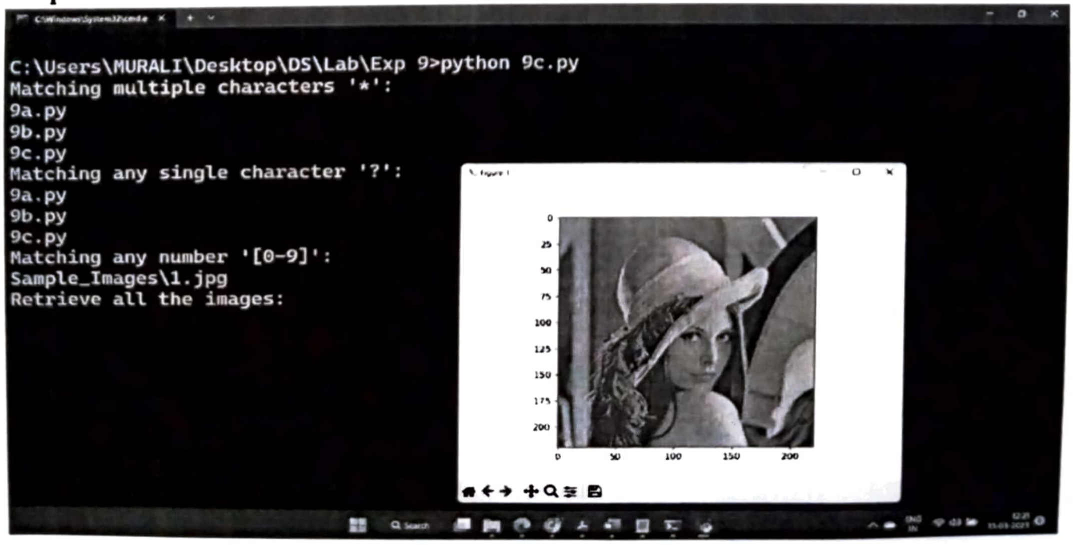
Multiple files using Glob:

- Python's Glob module lets you traverse through multiple files in the same location. Using glob.glob(), we can import all the files from our local folder that match a special pattern.
- These filename patterns can be made using different wildcards like
  "*" (for matching multiple characters),
  "?" (for matching any single character),
  '[0-9]' (for matching any number).

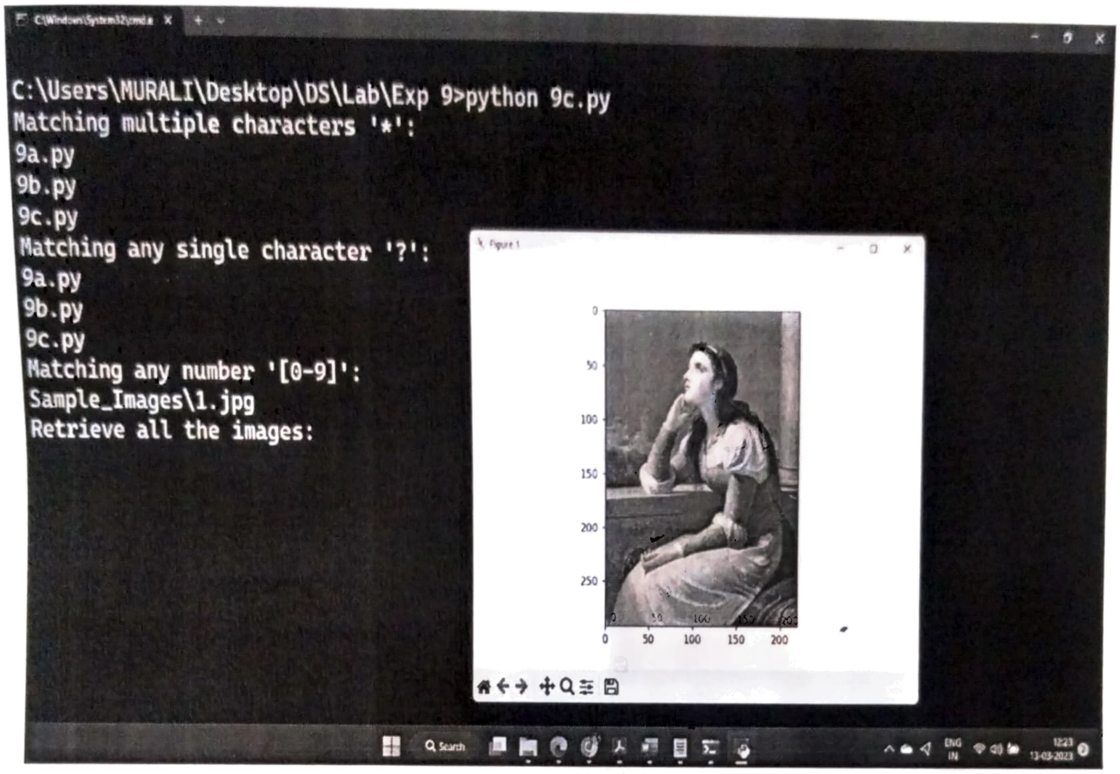
Example:

```
from PIL import Image
import glob2
import matplotlib.pyplot as plt
print("Matching multiple characters '*':")
for i in glob2.glob('*.py'):
    print(i)
print("Matching any single character '?':")
for i in glob2.glob('??.py'):
    print(i)
print("Matching any number '[0-9]':")
for i in glob2.glob('Sample_Images\[0-9].jpg'):
    print(i)
print("Retrieve all the images:")
images = glob2.glob('Sample_images\*.jpg')
for i in images[:3]:
    im = Image.open(i)
    plt.imshow(im)
    plt.show()
```

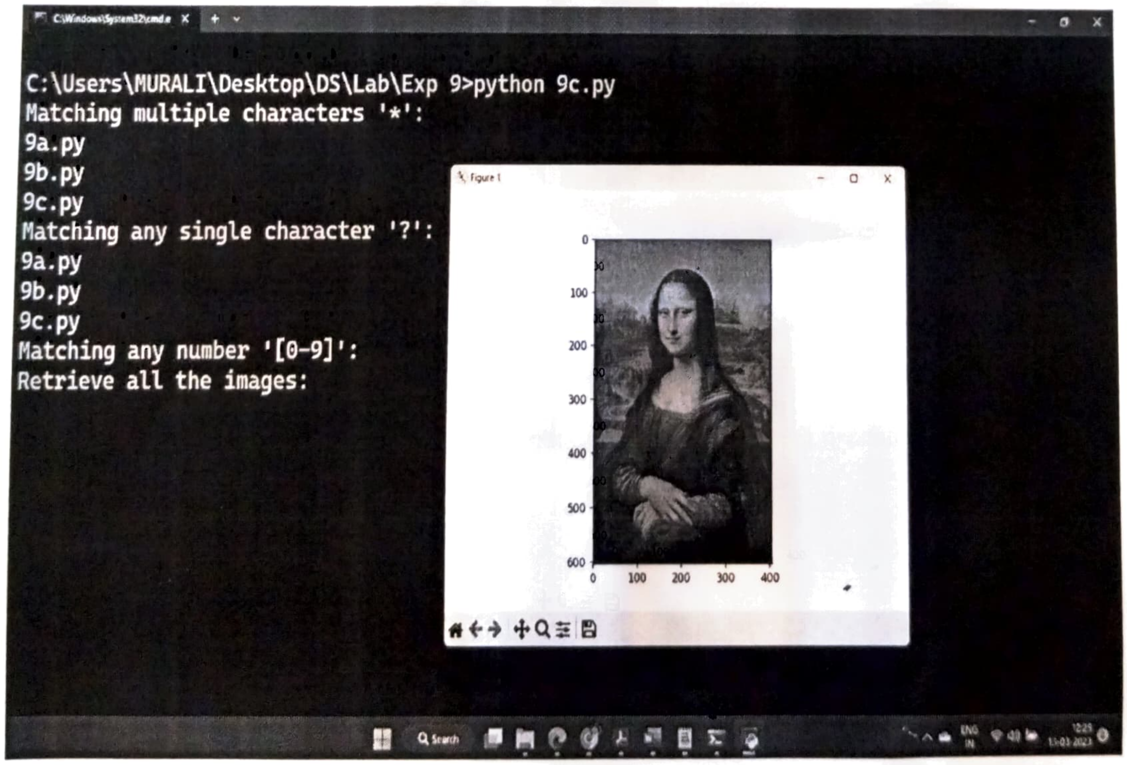
Output:



```
C:\Users\MURALI\Desktop\DS\Lab\Exp 9>python 9c.py
Matching multiple characters '*':
9a.py
9b.py
9c.py
Matching any single character '?':
9a.py
9b.py
9c.py
Matching any number '[0-9]':
Sample_Images\1.jpg
Retrieve all the images:
```



```
C:\Users\MURALI\Desktop\DS\Lab\Exp 9>python 9c.py
Matching multiple characters '*':
9a.py
9b.py
9c.py
Matching any single character '?':
9a.py
9b.py
9c.py
Matching any number '[0-9]':
Sample_Images\1.jpg
Retrieve all the images:
```



```
C:\Users\MURALI\Desktop\DS\Lab\Exp 9>python 9c.py
Matching multiple characters '*':
9a.py
9b.py
9c.py
Matching any single character '?':
9a.py
9b.py
9c.py
Matching any number '[0-9]':
Retrieve all the images:
```

9 (D):

Importing data from database:

Data in databases is stored in the form of tables and these systems are known as Relational database management systems (RDBMS). However, connecting to RDBMS and retrieving the data from it can prove to be quite a challenging task. we can easily do this using Python's built-in modules!

One of the most popular RDBMS is SQLite. It has many plus points:

- Lightweight database and hence it is easy to use in embedded software
- 35% faster reading and writing compared to the File System
- No intermediary server required. Reading and writing are done directly from the database files on the disk
- Cross-platform database file format. This means a file written on one machine can be copied to and used on a different machine with a different architecture
- There are many more reasons for its popularity. But for now, let's connect with an SQLite database and retrieve our data!

You will need to import the sqlite3 module to use SQLite. Then, you need to work through the following steps to access your data:

- Create a connection with the database connect(). You need to pass the name of your database to access it. It returns a Connection object
- Once you have done that, you need to create a cursor object using the cursor() function. This will allow you to implement SQL commands with which you can manipulate your data
- You can execute the commands in SQL by calling the execute() function on the cursor object. Since we are retrieving data from the database, we will use the SELECT statement and store the query in an object
- Store the data from the object into a dataframe by either calling fetchone(), for one row, or fecthall(), for all the rows, function on the object

And just like that, you have retrieved the data from the database into a Pandas dataframe!

A good practice is to save/commit your transactions using the commit() function even if you are only reading the data.

Example:

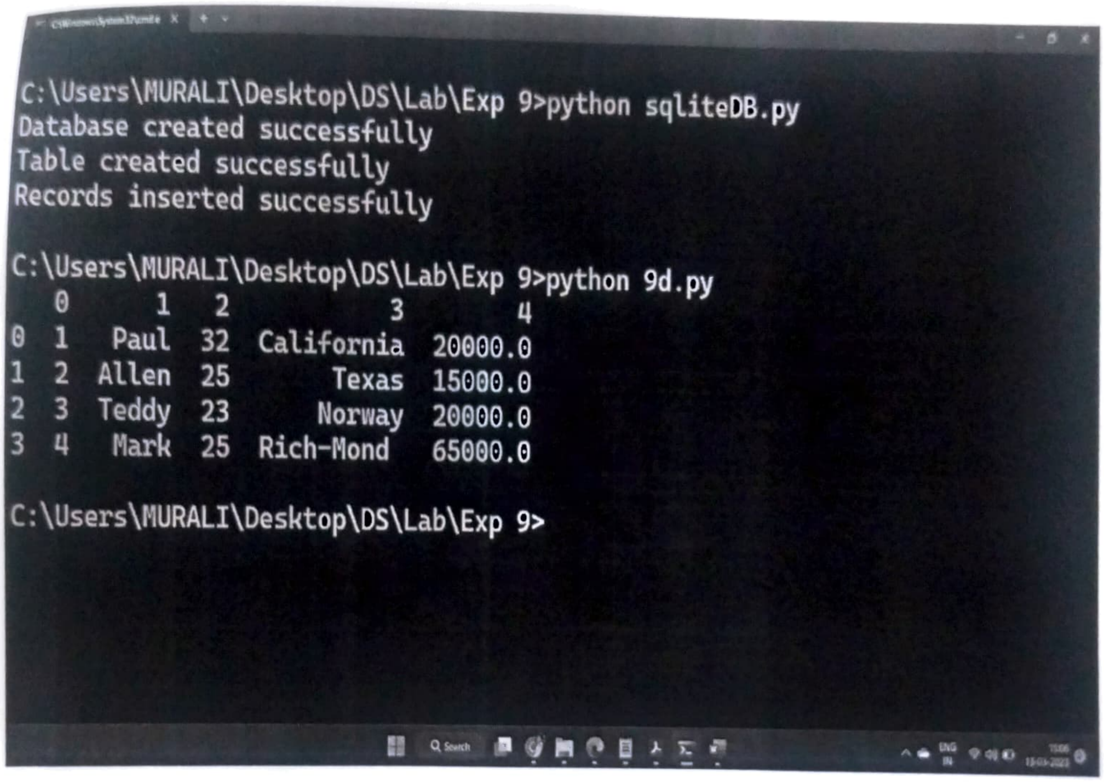
SqliteDB.py

```
import sqlite3
conn = sqlite3.connect('myDB.db')
print("Database created successfully")

conn.execute('''CREATE TABLE COMPANY
      (ID INT PRIMARY KEY    NOT NULL,
      NAME        TEXT   NOT NULL,
      AGE         INT    NOT NULL,
      ADDRESS       CHAR(50),
      SALARY        REAL);''')
print("Table created successfully")

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
    VALUES (1, 'Paul', 32, 'California', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
    VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
    VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
    VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()
print("Records inserted successfully")
conn.close()
```

9d.py

```
import pandas as pd
import sqlite3

con=sqlite3.connect('myDB.db')
cur = con.cursor()
rs = cur.execute('select * from COMPANY')
df = pd.DataFrame(rs.fetchall())
con.commit()
print(df)
```

Output:



```
C:\Users\MURALI\Desktop\DS\Lab\Exp 9>python sqliteDB.py
Database created successfully
Table created successfully
Records inserted successfully

C:\Users\MURALI\Desktop\DS\Lab\Exp 9>python 9d.py
   0      1   2           3        4
0  1   Paul  32  California  20000.0
1  2  Allen  25       Texas  15000.0
2  3  Teddy  23      Norway  20000.0
3  4   Mark  25  Rich-Mond   65000.0

C:\Users\MURALI\Desktop\DS\Lab\Exp 9>
```