VIDHI ROHIRA

SY BTECH

COMPUTER ENGINEERING

PD LAB-2

231071052

PD LAB 2

I] AIM:- To create contact application using dictionary and tkinkter.

II] THEORY:-

TKINKTER:- Tkinter is a Python library for creating graphical user interfaces (GUIs). It wraps the Tcl/Tk toolkit and offers a range of features:

- **Widgets**: Components like labels, buttons, and text boxes.
- **Layout Management**: Tools like pack, grid, and place for arranging widgets.
- **Event Handling**: Bind functions to user actions like clicks and key presses.
- **Geometry Management**: Control the size and placement of windows and widgets.
- **Customization**: Modify the appearance and behavior of widgets, such as colors and fonts.

It’s included with Python and is ideal for building desktop applications.

DICTIONARY:- In Python, a dictionary is a built-in data type used to store collections of key-value pairs. Each key in a dictionary maps to a specific value, allowing for fast retrieval based on the key. Dictionaries are mutable, meaning you can change their contents after creation, and they are unordered, meaning the items do not have a specific sequence.

Here's a brief overview:

- **Syntax**: Dictionaries are created using curly braces {} with key-value pairs separated by colons. For example: my_dict = {'name': 'Alice', 'age': 25}.
- **Keys and Values**: Keys must be unique and immutable (e.g., strings, numbers, tuples), while values can be of any data type and can be duplicated.
- **Accessing Values**: Values are accessed using their corresponding keys. For example: my_dict['name'] would return 'Alice'.
- **Common Methods**: Some common dictionary methods include .get(), .keys(), .values(), and .items().

Q] WHAT IS CONTACT APPLICATION?

A contact application is a software program designed to manage and organize contact information. It typically allows users to store, retrieve, and manage details about individuals or organizations. Common features include:

- Contact Storage: Users can save various details such as names, phone numbers, email addresses, and physical addresses.

- Search and Filter: Functions to search for and filter contacts based on different criteria.

- Groups and Categories: Ability to organize contacts into groups or categories for easier management.

- Import and Export: Options to import contacts from other sources or export them for backup or use in other applications.

- Syncing: Integration with other services (like email or calendar applications) to keep contact information synchronized across devices.

Examples include built-in contact managers on smartphones, email applications like Outlook or Gmail, and dedicated contact management software for personal or business use.

Q] HOW CAN WE MAKE A CONTACT APPLICATION USING DICTIONARY AND TKINKTER ?

To create a contact application using a dictionary and Tkinter, you would start by using a dictionary to store contact information, where each key represents a contact name and the corresponding value is another dictionary with details like phone number and email. With Tkinter, you can build a graphical user interface that includes fields for inputting and displaying contact information, such as text entries for names, phone numbers, and email addresses, and buttons for adding, deleting, and searching contacts. For example, when a user adds a contact, the application updates the dictionary and refreshes the displayed list of contacts in the GUI. You can also use Tkinter's widgets to implement features like searching and filtering contacts. This approach combines the dictionary's efficient data management with Tkinter's ability to create user-friendly interfaces.

III] CODE AND OUTPUT :-

```
import tkinter as tk
from tkinter import messagebox
import re  # Regular expressions for validation

contacts_db = {}
users_db = {"admin": "password123"}  # Example user database for login

def show_frame(frame):
    frame_login.pack_forget()
    frame_add_user.pack_forget()
    frame_search_user.pack_forget()
    frame.pack(fill='both', expand=True)

def validate_phone(phone):
    """Validate phone number: must be digits and 10 characters long."""
    return phone.isdigit() and len(phone) == 10

def validate_email(email):
    """Validate email format."""
    email_regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(email_regex, email) is not None

def save_contact():
    username = entry_username.get()
    phone = entry_phone.get()
    email = entry_email.get()

    if not username or not phone or not email:
        messagebox.showwarning("Input Error", "All fields are required.")
        return

    if not validate_phone(phone):
        messagebox.showwarning("Input Error", "Phone number must be 10 digits
long.")
        return

    if not validate_email(email):
        messagebox.showwarning("Input Error", "Invalid email format.")
        return

    if username in contacts_db:
        messagebox.showwarning("Duplicate User", "User already exists. Please
use a different username.")
        return
```

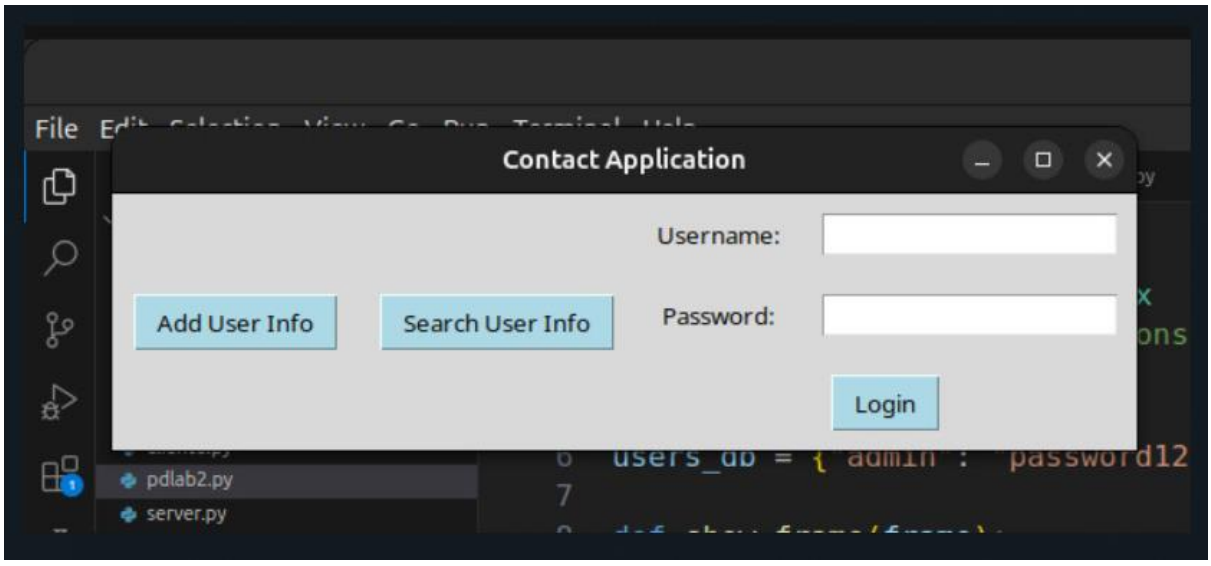
```
    contacts_db[username] = {
        "phone": phone,
        "email": email
    }

    messagebox.showinfo("Success", "Contact saved successfully!")
    update_contact_list()

    entry_username.delete(0, tk.END)
    entry_phone.delete(0, tk.END)
    entry_email.delete(0, tk.END)

def update_contact_list():
    """Update the Listbox with the current contacts."""
    listbox_contacts.delete(0, tk.END)
    for username, details in contacts_db.items():
        contact_info = f"Username: {username}, Phone: {details['phone']},
Email: {details['email']}"
        listbox_contacts.insert(tk.END, contact_info)

def delete_contact():
    selected_index = listbox_contacts.curselection()
    if not selected_index:
        messagebox.showwarning("Selection Error", "Please select a contact to
delete.")
        return

    contact_info = listbox_contacts.get(selected_index)
    username = contact_info.split(',')[0].split(': ')[1]

    if username in contacts_db:
        del contacts_db[username]
        messagebox.showinfo("Success", "Contact deleted successfully!")
        update_contact_list()

def edit_contact():
    selected_index = listbox_contacts.curselection()
    if not selected_index:
        messagebox.showwarning("Selection Error", "Please select a contact to
edit.")
        return

    contact_info = listbox_contacts.get(selected_index)
    username = contact_info.split(',')[0].split(': ')[1]

    if username in contacts_db:
        # Populate the entry fields with the contact details
        entry_username.delete(0, tk.END)
```

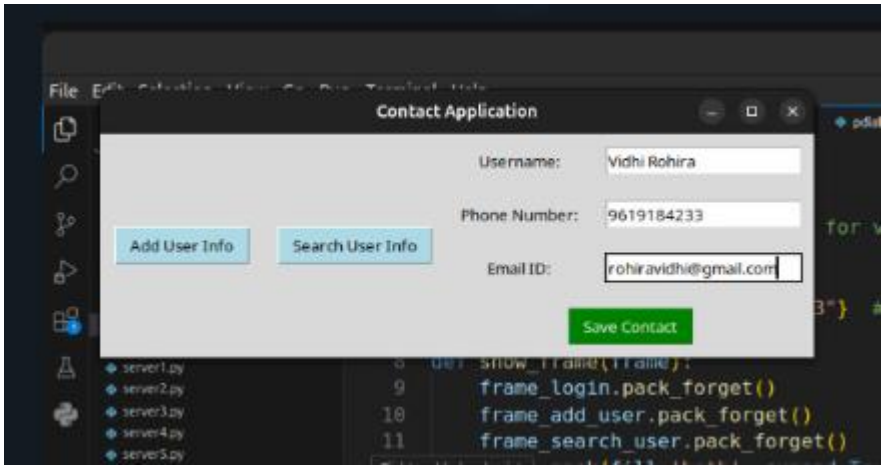
```
        entry_username.insert(0, username)
        entry_phone.delete(0, tk.END)
        entry_phone.insert(0, contacts_db[username]['phone'])
        entry_email.delete(0, tk.END)
        entry_email.insert(0, contacts_db[username]['email'])

        # Change the Save Contact button to an Update Contact button
        save_button.config(text="Update Contact", command=lambda:
update_contact(username))

def update_contact(original_username):
    username = entry_username.get()
    phone = entry_phone.get()
    email = entry_email.get()

    if not username or not phone or not email:
        messagebox.showwarning("Input Error", "All fields are required.")
        return

    if not validate_phone(phone):
        messagebox.showwarning("Input Error", "Phone number must be 10 digits
long.")
        return

    if not validate_email(email):
        messagebox.showwarning("Input Error", "Invalid email format.")
        return

    if username != original_username and username in contacts_db:
        messagebox.showwarning("Duplicate User", "User already exists. Please
use a different username.")
        return

    # Update the contact information
    contacts_db.pop(original_username)  # Remove the old contact
    contacts_db[username] = {
        "phone": phone,
        "email": email
    }

    messagebox.showinfo("Success", "Contact updated successfully!")
    update_contact_list()

    entry_username.delete(0, tk.END)
    entry_phone.delete(0, tk.END)
    entry_email.delete(0, tk.END)

    # Reset the button to Save Contact
```

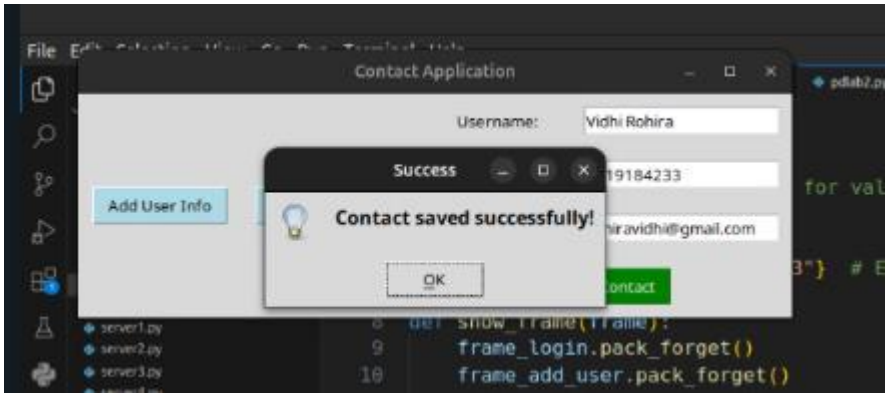
```
    save_button.config(text="Save Contact", command=save_contact)

def login():
    username = entry_login_username.get()
    password = entry_login_password.get()

    if username in users_db and users_db[username] == password:
        messagebox.showinfo("Login Success", "Welcome!")
        show_frame(frame_add_user)
    else:
        messagebox.showerror("Login Error", "Invalid username or password.")

root = tk.Tk()
root.title("Contact Application")

# Frames for different parts of the application
frame_login = tk.Frame(root)
frame_add_user = tk.Frame(root)
frame_search_user = tk.Frame(root)

# Login frame
tk.Label(frame_login, text="Username:").grid(row=0, column=0, padx=10,
pady=10)
entry_login_username = tk.Entry(frame_login)
entry_login_username.grid(row=0, column=1, padx=10, pady=10)

tk.Label(frame_login, text="Password:").grid(row=1, column=0, padx=10,
pady=10)
entry_login_password = tk.Entry(frame_login, show='*')
entry_login_password.grid(row=1, column=1, padx=10, pady=10)

tk.Button(frame_login, text="Login", command=login,
bg='lightblue').grid(row=2, columnspan=2, pady=10)

# Add User Info frame
tk.Label(frame_add_user, text="Username:").grid(row=0, column=0, padx=10,
pady=10)
entry_username = tk.Entry(frame_add_user)
entry_username.grid(row=0, column=1, padx=10, pady=10)

tk.Label(frame_add_user, text="Phone Number:").grid(row=1, column=0, padx=10,
pady=10)
entry_phone = tk.Entry(frame_add_user)
entry_phone.grid(row=1, column=1, padx=10, pady=10)

tk.Label(frame_add_user, text="Email ID:").grid(row=2, column=0, padx=10,
pady=10)
entry_email = tk.Entry(frame_add_user)
```

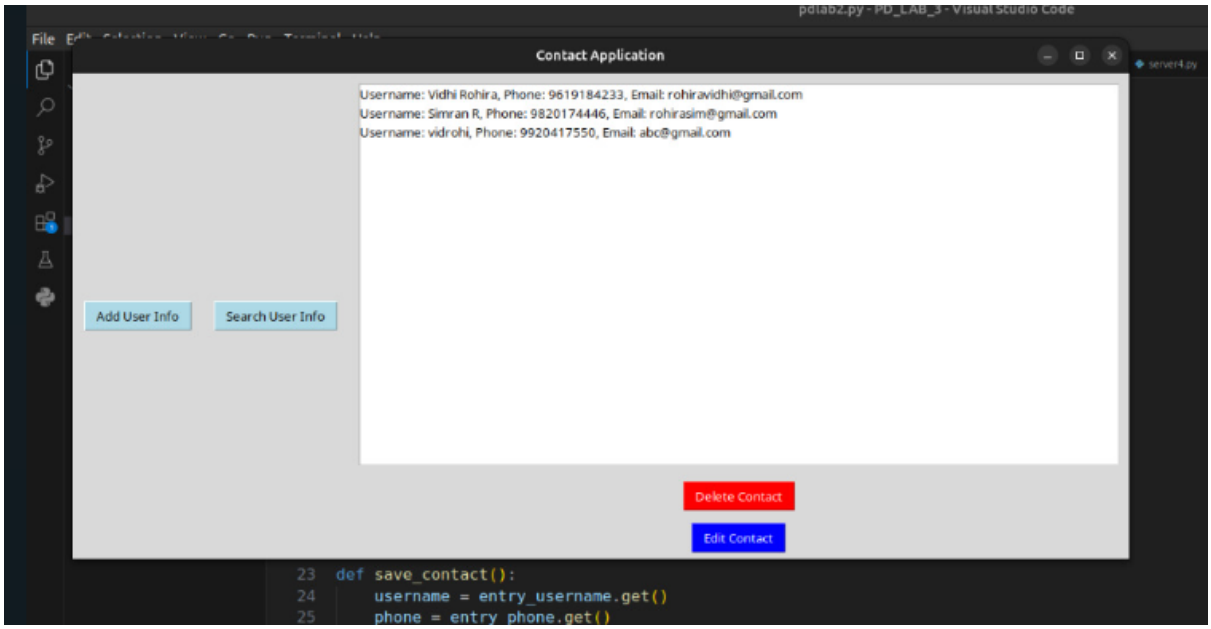
```
entry_email.grid(row=2, column=1, padx=10, pady=10)

# Save button that changes to Update button in edit mode with color
save_button = tk.Button(frame_add_user, text="Save Contact",
command=save_contact, bg='green', fg='white')
save_button.grid(row=3, columnspan=2, pady=10)

# Search User Info frame
listbox_contacts = tk.Listbox(frame_search_user, width=100, height=20)
listbox_contacts.pack(padx=10, pady=10)

# Delete and Edit buttons with color
tk.Button(frame_search_user, text="Delete Contact", command=delete_contact,
bg='red', fg='white').pack(pady=5)
tk.Button(frame_search_user, text="Edit Contact", command=edit_contact,
bg='blue', fg='white').pack(pady=6)

# Navigation buttons
tk.Button(root, text="Add User Info", command=lambda:
show_frame(frame_add_user), bg='lightblue').pack(side='left', padx=11,
pady=10)
tk.Button(root, text="Search User Info", command=lambda:
show_frame(frame_search_user), bg='lightblue').pack(side='left', padx=11,
pady=10)

# Show the login frame initially
show_frame(frame_login)

root.mainloop()
```

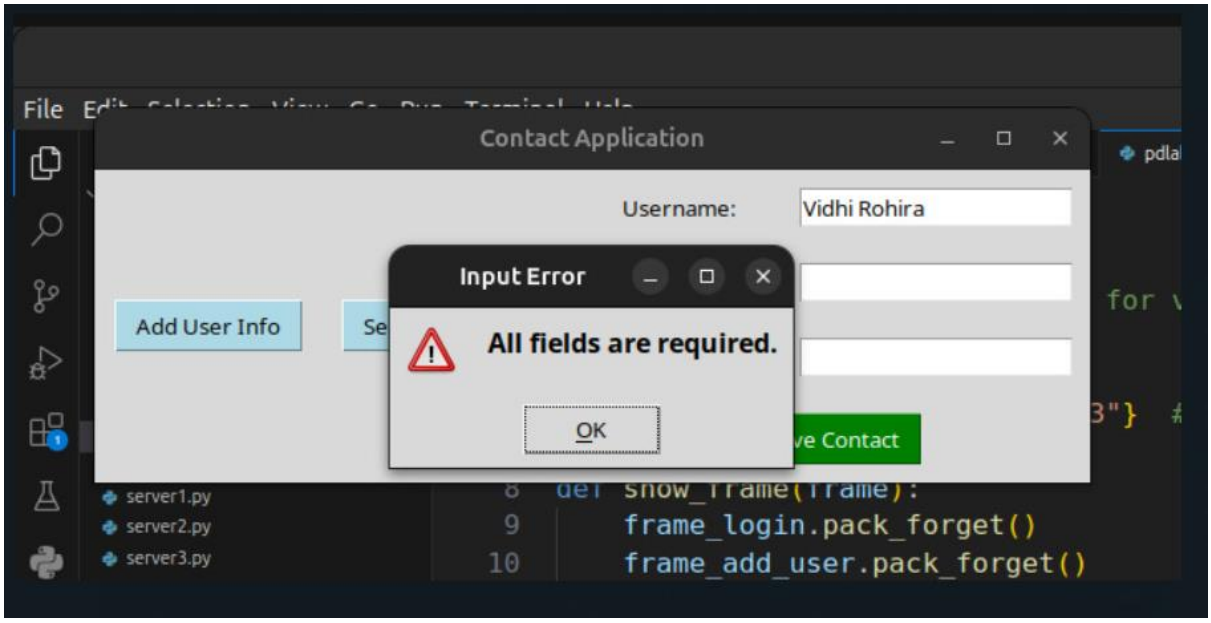
OUTPUT:-

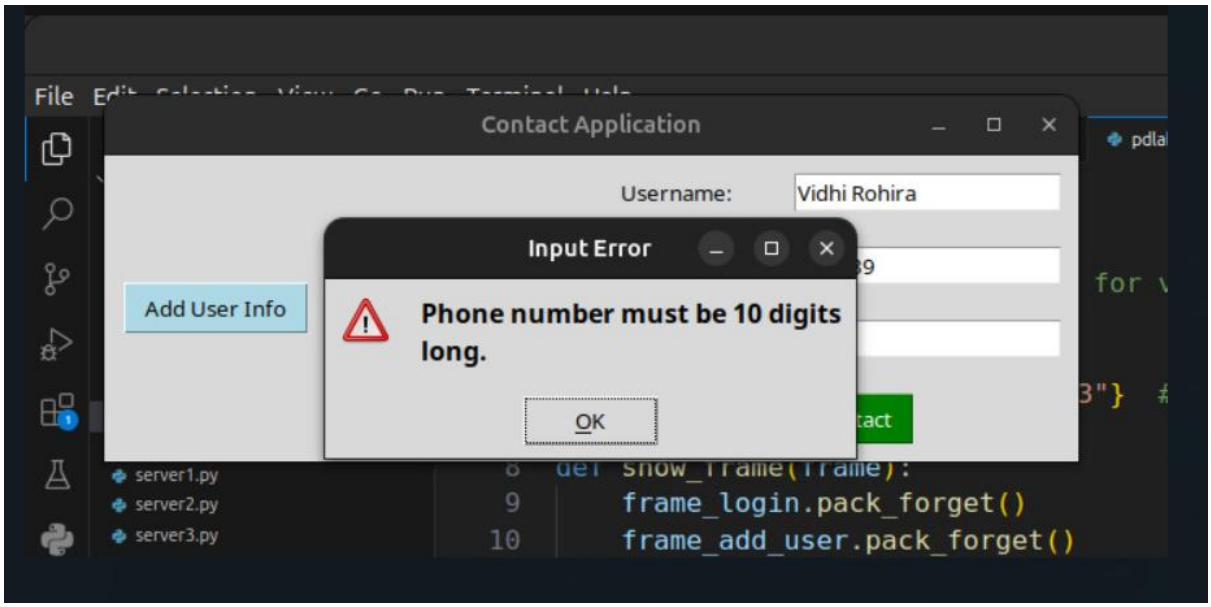


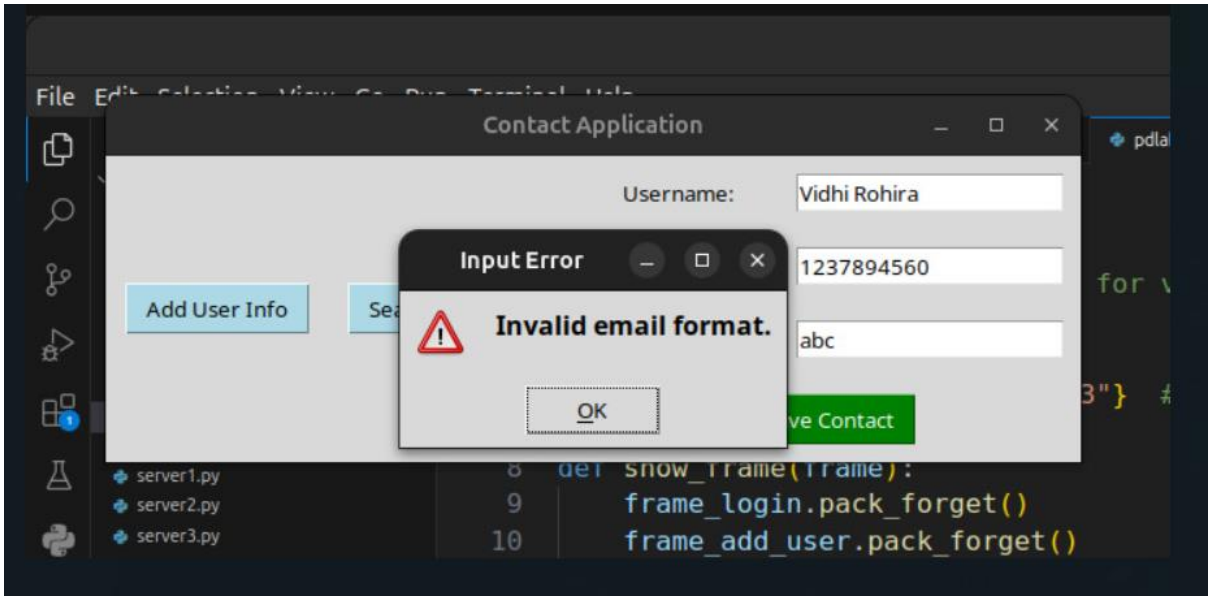

Contact Application
Username:
Vidhi Rohira
Phone Number:
9619184233
Email ID:
rohiravidhi@gmail.com
Add User Info
Search User Info
Save Contact
frame_login.pack_forget()
frame_add_user.pack_forget()
frame_search_user.pack_forget()

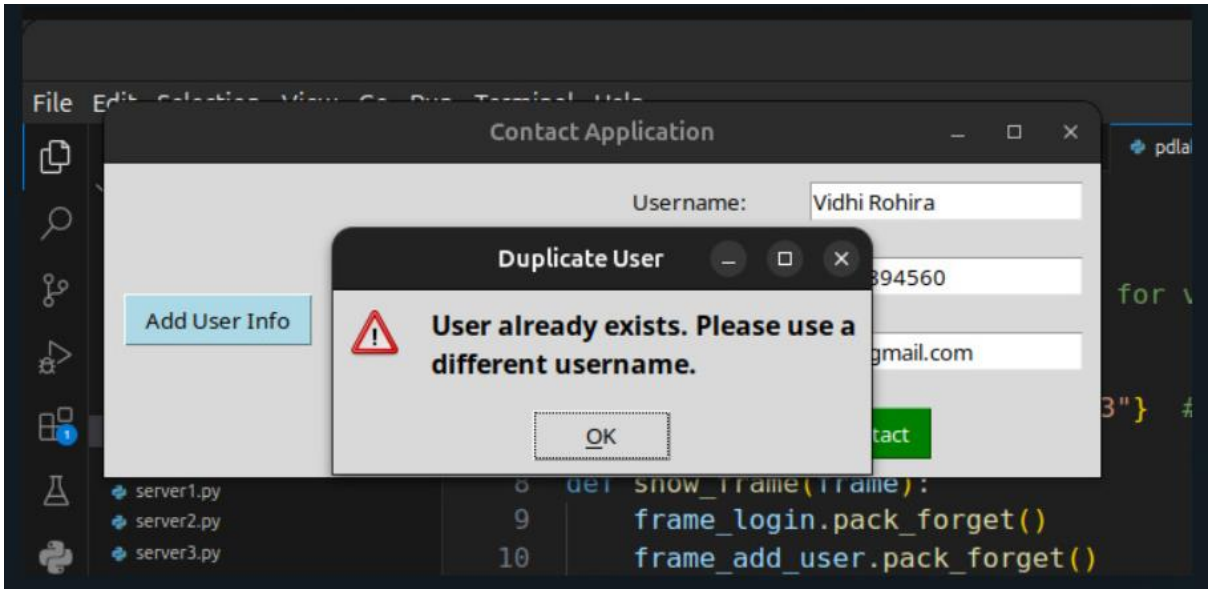

Contact Application
Username:
Vidhi Rohira
Success
Contact saved successfully!
OK
Add User Info
frame_login.pack_forget()
frame_add_user.pack_forget()

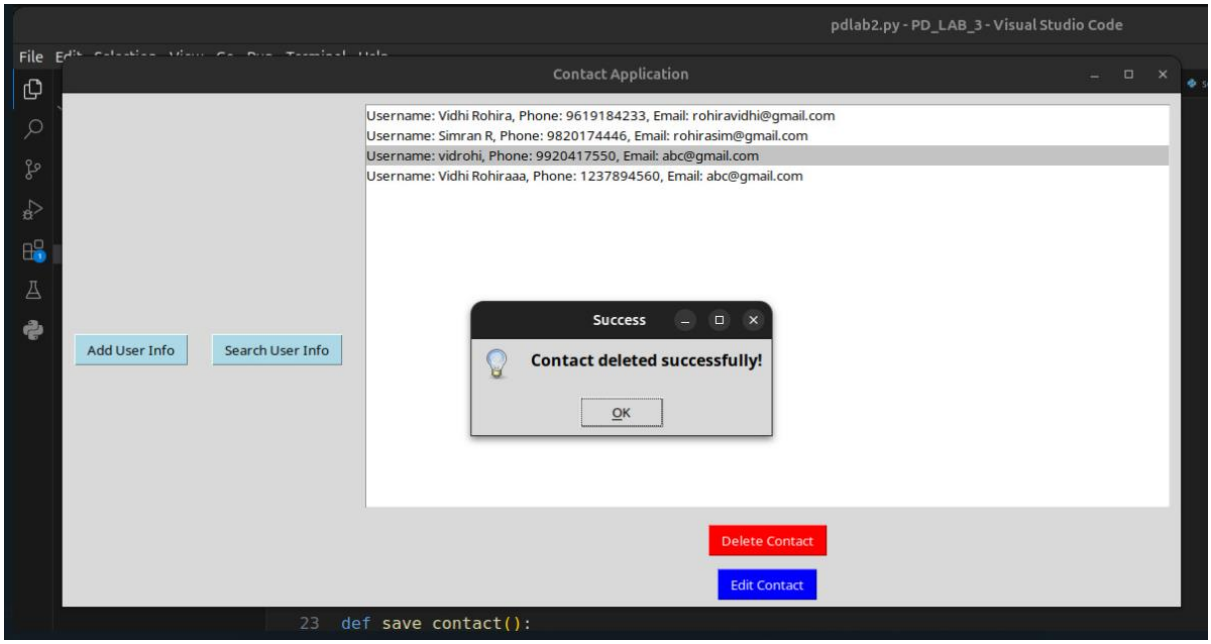

Contact Application
Username: Vidhi Rohira, Phone: 9619184233, Email: rohiravidhi@gmail.com
Username: Simran R, Phone: 9820174446, Email: rohirasim@gmail.com
Username: vidrohi, Phone: 9920417550, Email: abc@gmail.com
Add User Info
Search User Info
Delete Contact
Edit Contact
def save_contact():
username = entry_username.get()
phone = entry_phone.get()


File
Contact Application
Username:
Vidhi Rohira
Input Error
All fields are required.
OK
Add User Info
server1.py
server2.py
server3.py
def show_frame(frame):
frame_login.pack_forget()
frame_add_user.pack_forget()


File
Contact Application
Username:
Vidhi Rohira
Input Error
Phone number must be 10 digits long.
OK
Add User Info
server1.py
server2.py
server3.py
def show_frame(frame):
frame_login.pack_forget()
frame_add_user.pack_forget()


File
Contact Application
Username:
Vidhi Rohira
1237894560
abc
Add User Info
Input Error
Invalid email format.
OK
ve Contact
server1.py
server2.py
server3.py
def show_frame(frame):
frame_login.pack_forget()
frame_add_user.pack_forget()


File
Contact Application
Username:
Vidhi Rohira
Add User Info
Duplicate User
User already exists. Please use a different username.
OK
server1.py
server2.py
server3.py
def show_frame(frame):
frame_login.pack_forget()
frame_add_user.pack_forget()



IV] CONCLUSION:- Hence we have studied about tkinkter and dictionary in detail. Had a look at how we can use tkinkter and dictionary to create a contact application. Seen what a contact application is and implemented the code for the same.