CREATE GUI APPLICATION USING PYTHON TKINTER

TEAM 1

TEAM MEMBERS

- AIE-02 ANEESH KRISHNA S
- AIE-15 SRI HARSHINI S
- AIE-16 SUDHISHNA S
- AIE-22 VISHAL BHARATH R
- AIE-23 VIKRAM P
- AIE-32 RAGESH R
- AIE-34 SHALINI V

AIM:

Aim of the above program is to provide a simple and interactive contact management system with a graphical user interface (GUI) using the Tkinter library in Python

OBJECTIVE:

The objective of the above program is to create a simple contact management system using a graphical user interface (GUI) with Tkinter, which is a standard GUI toolkit for Python. The program allows users to perform the following actions:

✓Add a Contact:

Users can input details such as name, email, father's name, phone number, and date of birth.

The program validates the input to ensure that the entered data meets certain criteria (e.g., alphabetic characters for names, correct email format, 10-digit phone number, and a valid date of birth format).

✓View Contact:

Users can select a contact from the list and view detailed information about that contact, including name, email, father's name, phone number, and date of birth.

✓Edit Contact:

Users can select a contact from the list, and the program populates the input fields with the contact's information.

Users can make changes and click the "Edit Contact" button to update the contact details.

The edited contact is saved, and the contact list is updated.

✓Delete Contact:

Users can select a contact from the list and click the "Delete Contact" button to remove the contact.

The contact is deleted from the list, and the updated list is saved.

✓Load and Save Contacts:

Contacts are saved to a file ("contacts.txt") so that they persist between program executions.

When the program starts, it loads existing contacts from the file.

✓GUI Interaction:

The program provides a graphical user interface with input fields, buttons, and a list box to display contacts.

Tkinter is used to create the GUI elements and handle user interactions.

Overall, the program serves as a basic example of a contact management system, demonstrating how to use Tkinter for GUI development in Python and incorporating input validation to ensure data integrity.

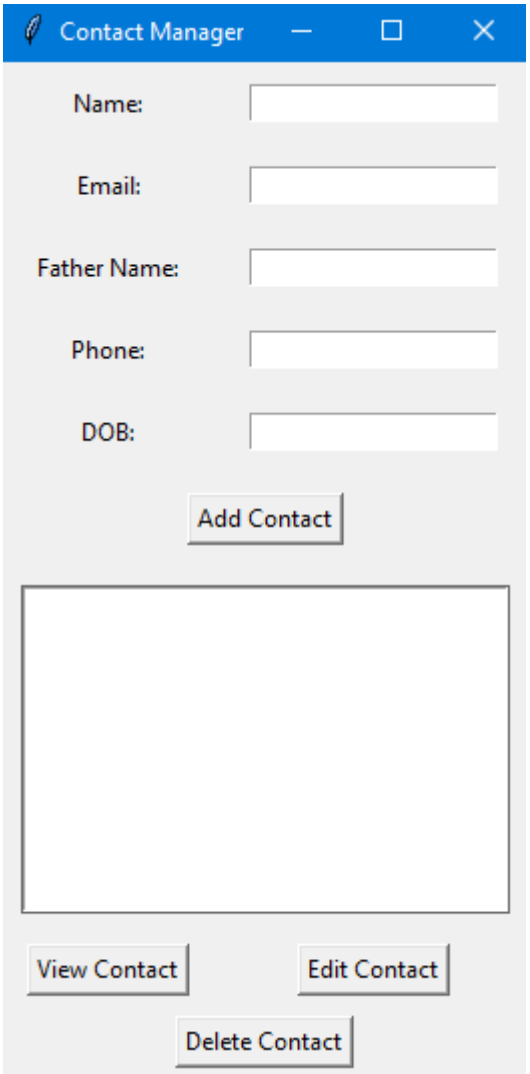
CODING:

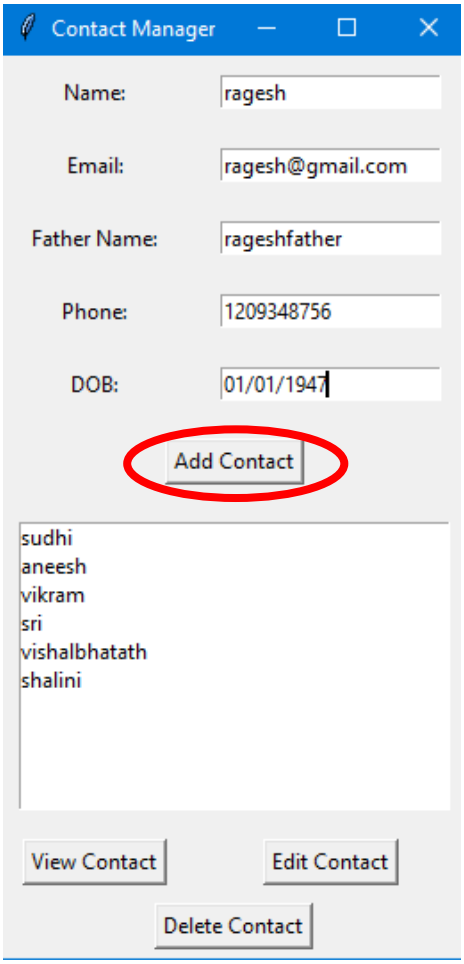
```
import tkinter as tk
from tkinter import messagebox

class Contact:
    def __init__(self, name, email, father_name, phone, dob):
        self.name = name
        self.email = email
        self.father_name = father_name
        self.phone = phone
        self.dob = dob

class ContactManager:
    def __init__(self, master):
        self.master = master
        self.master.title("Contact Manager")

        self.contacts = []
        self.load_contacts()

        self.name_var = tk.StringVar()
        self.email_var = tk.StringVar()
        self.father_name_var = tk.StringVar()
        self.phone_var = tk.StringVar()
        self.dob_var = tk.StringVar()

        self.create_widgets()

    def create_widgets(self):
        tk.Label(self.master, text="Name:").grid(row=0, column=0, padx=10, pady=10)
        tk.Entry(self.master, textvariable=self.name_var).grid(row=0, column=1, padx=10,
pady=10)

        tk.Label(self.master, text="Email:").grid(row=1, column=0, padx=10, pady=10)
        tk.Entry(self.master, textvariable=self.email_var).grid(row=1, column=1, padx=10,
pady=10)

        tk.Label(self.master, text="Father Name:").grid(row=2, column=0, padx=10,
pady=10)
        tk.Entry(self.master, textvariable=self.father_name_var).grid(row=2, column=1,
padx=10, pady=10)

        tk.Label(self.master, text="Phone:").grid(row=3, column=0, padx=10, pady=10)
        tk.Entry(self.master, textvariable=self.phone_var).grid(row=3, column=1, padx=10,
pady=10)

        tk.Label(self.master, text="DOB:").grid(row=4, column=0, padx=10, pady=10)
        tk.Entry(self.master, textvariable=self.dob_var).grid(row=4, column=1, padx=10,
pady=10)

        tk.Button(self.master, text="Add Contact", command=self.add_contact).grid(row=5,
column=0, columnspan=2, pady=10)

        self.contact_listbox = tk.Listbox(self.master, selectmode=tk.SINGLE, height=10,
width=40)
        self.contact_listbox.grid(row=6, column=0, columnspan=2, padx=10, pady=10)

        tk.Button(self.master, text="View Contact",
command=self.view_contact).grid(row=7, column=0, pady=5)
        tk.Button(self.master, text="Edit Contact",
command=self.edit_contact).grid(row=7, column=1, pady=5)
        tk.Button(self.master, text="Delete Contact",
command=self.delete_contact).grid(row=8, column=0, columnspan=2, pady=5)

        self.populate_contact_listbox()
```

```
    def add_contact(self):
        name = self.name_var.get()
        email = self.email_var.get()
        father_name = self.father_name_var.get()
        phone = self.phone_var.get()
        dob = self.dob_var.get()

        if self.validate_input(name, email, father_name, phone, dob):
            contact = Contact(name, email, father_name, phone, dob)
            self.contacts.append(contact)
            self.save_contacts()
            self.populate_contact_listbox()
            self.clear_input_fields()
        else:
            messagebox.showwarning("Input Error", "Please check your entries and make
sure they meet the requirements.")

    def validate_input(self, name, email, father_name, phone, dob):
        if not name.isalpha() or not father_name.isalpha():
            messagebox.showwarning("Input Error", "Name and Father Name should contain
only alphabetic characters.")
            return False

        if not email.endswith("@gmail.com"):
            messagebox.showwarning("Input Error", "Email address must end with
@gmail.com.")
            return False

        if not phone.isdigit() or len(phone) != 10:
            messagebox.showwarning("Input Error", "Phone number should be a 10-digit
number.")
            return False

        try:
            day, month, year = map(int, dob.split('/'))
            if not (1 <= day <= 31 and 1 <= month <= 12 and 1900 <= year <= 9999):
                raise ValueError()
        except ValueError:
            messagebox.showwarning("Input Error", "Invalid date of birth. Please use the
format dd/mm/yyyy.")
            return False

        return True

    def view_contact(self):
        selected_index = self.contact_listbox.curselection()
        if selected_index:
            contact = self.contacts[selected_index[0]]
            messagebox.showinfo("Contact Information", f"Name: {contact.name}\nEmail:
{contact.email}\nFather Name: {contact.father_name}\nPhone: {contact.phone}\nDOB:
{contact.dob}")
        else:
            messagebox.showwarning("Selection Error", "Please select a contact to view.")

    def edit_contact(self):
        selected_index = self.contact_listbox.curselection()
        if selected_index:
            contact = self.contacts[selected_index[0]]
            self.name_var.set(contact.name)
            self.email_var.set(contact.email)
            self.father_name_var.set(contact.father_name)
            self.phone_var.set(contact.phone)
            self.dob_var.set(contact.dob)

            self.contacts.pop(selected_index[0])
            self.save_contacts()
            self.populate_contact_listbox()
        else:
            messagebox.showwarning("Selection Error", "Please select a contact to edit.")
```

```
    def delete_contact(self):
        selected_index = self.contact_listbox.curselection()
        if selected_index:
            self.contacts.pop(selected_index[0])
            self.save_contacts()
            self.populate_contact_listbox()
            self.clear_input_fields()
        else:
            messagebox.showwarning("Selection Error", "Please select a contact to
delete.")

    def load_contacts(self):
        try:
            with open("contacts.txt", "r") as file:
                for line in file:
                    name, email, father_name, phone, dob = line.strip().split(',')
                    contact = Contact(name, email, father_name, phone, dob)
                    self.contacts.append(contact)
        except FileNotFoundError:
            pass

    def save_contacts(self):
        with open("contacts.txt", "w") as file:
            for contact in self.contacts:

file.write(f"{contact.name},{contact.email},{contact.father_name},{contact.phone},{contac
t.dob}\n")

    def populate_contact_listbox(self):
        self.contact_listbox.delete(0, tk.END)
        for contact in self.contacts:
            self.contact_listbox.insert(tk.END, contact.name)

    def clear_input_fields(self):
        self.name_var.set("")
        self.email_var.set("")
        self.father_name_var.set("")
        self.phone_var.set("")
        self.dob_var.set("")

if __name__ == "__main__":
    root = tk.Tk()
    contact_manager = ContactManager(root)
    root.mainloop()
```

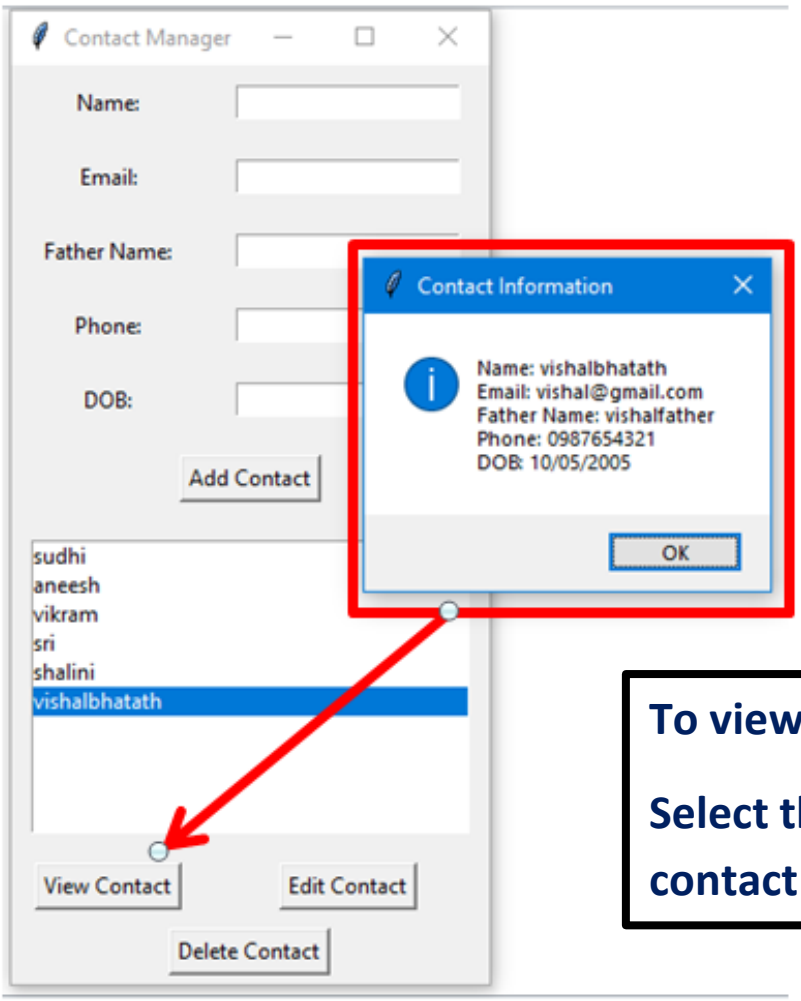
OUTPUT:


Contact Manager
Name:
Email:
Father Name:
Phone:
DOB:
Add Contact
View Contact
Edit Contact
Delete Contact



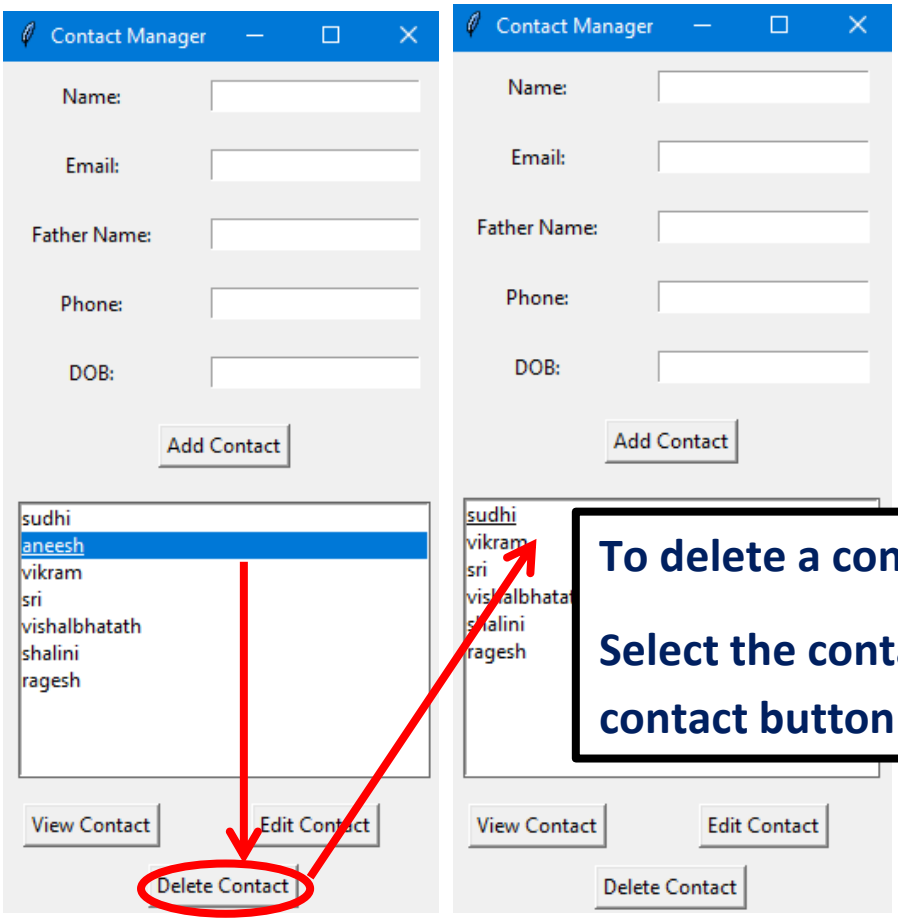
Type the required information

And click the Add Contact button



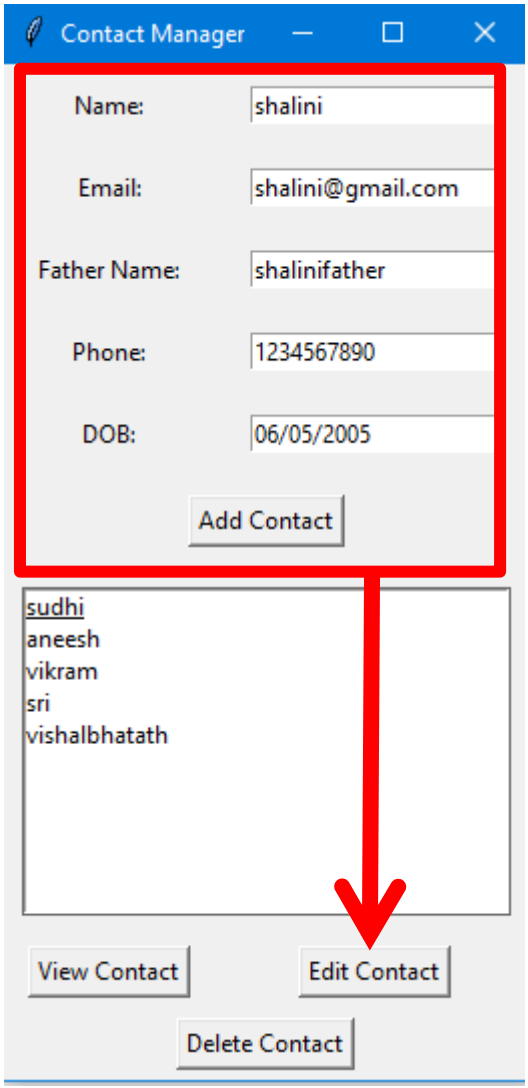
To view the contact details

Select the contact and Click view contact button



To delete a contact

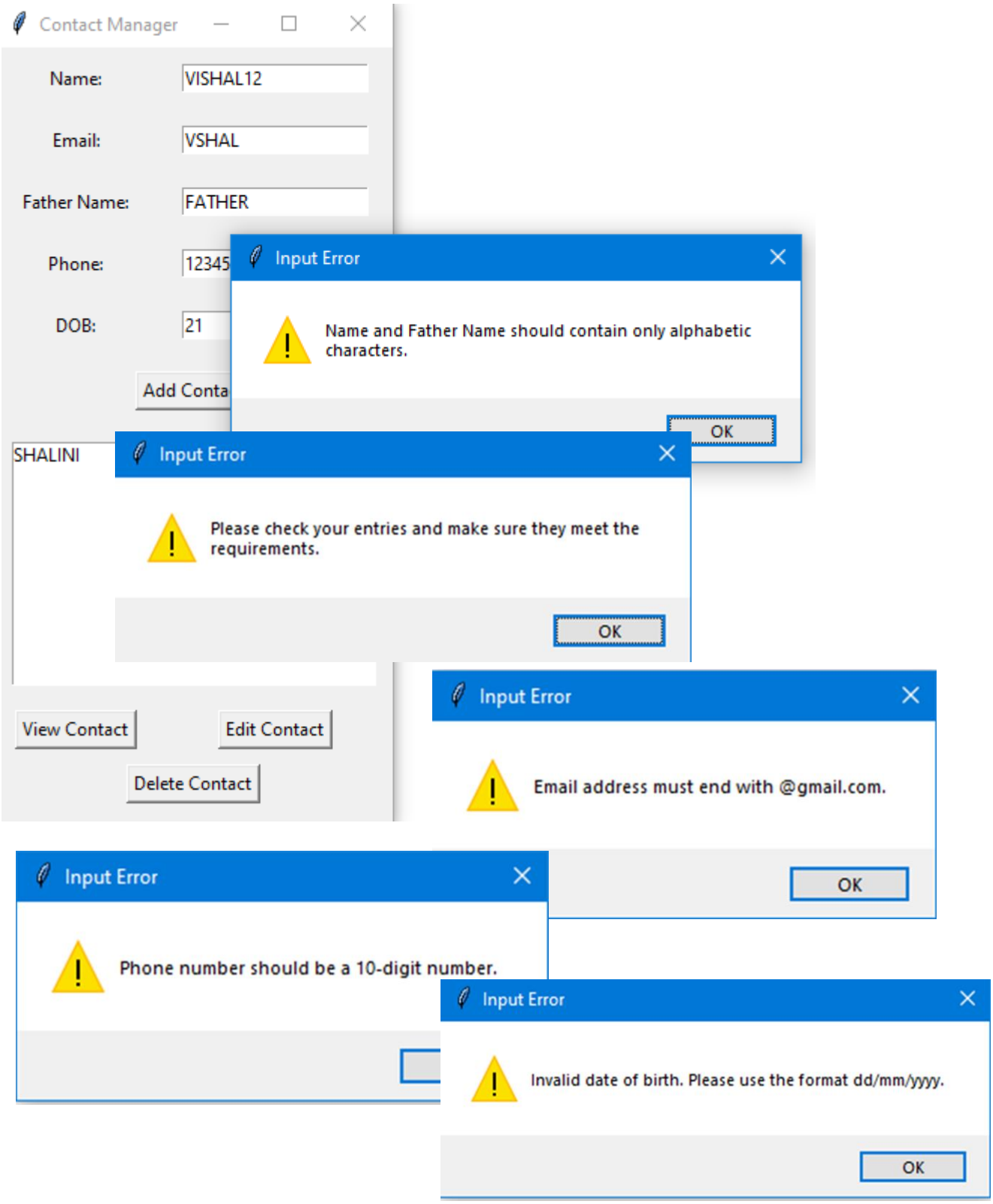
Select the contact and click delete contact button



To edit a contact

Select the contact and click Edit contact button and perform the required editing process

- ERROR MESSAGE DISPLAYED



Conclusion:

In conclusion, the provided Python program showcases a simple contact management system with a Tkinter-based GUI. Key features include contact addition, viewing, editing, and deletion, along with input validation for data integrity. Contacts persist between executions in a file ("contacts.txt"), and the program serves as an educational example for GUI development, input validation, and file handling in Python. The program offers a foundation for further development and learning.