PYTHON OPEN ENDED LAB

AIRPORT MANAGEMENT SYSTEM

SUBMITTED TO: SIR SIRTAJ AHMED MALIK

SLOT: (WEDNESDAY 8:30 – 11:30)

GROUP MEMBERS

○ NAMRA ABID	(62531)
○ MUHAMMAD ALYAN	(62958)
○ FAAIZ AHMED KHAN	(62549)
○ ALEENA QAMAR	(56110)
○ ROMAISA SHAHBAZ	(56056)

AIRPORT MANAGEMENT SYSTEM

Objective: To create a graphical user interface (GUI) application for managing flight details, allowing users to view, update, search, and add flight information.

Motivation: The motivation behind this project is to simplify the management of flight data, providing an easy-to-use platform for tracking flight statuses, schedules, and passenger details.

Concept: The application uses Python's tkinter library to build a GUI-based flight management system. It incorporates features like flight searching, data entry, and status updates to streamline flight data management**.**

Problem Statement: Efficient management of flight details and statuses is crucial for operations. A user-friendly application is needed to handle tasks such as searching for flights, viewing all available flights, updating flight statuses, and adding new flights.

✓ Design / Ways & Means:

- **Introduction and Requirements:**
  The system should allow users to interact with flight data through a GUI, providing functionalities such as flight search, viewing all flights, updating flight statuses, and adding new flights.
- **Data Structure Selection:**
  A Python dictionary is used to store flight details, with the flight number as the key and a nested dictionary containing time, destination, and status as the values.
- **Basic Implementation:**
  The tkinter library is used to design the GUI. Functions are implemented to handle user interactions, including searching for flights, resetting fields, viewing all flights, updating flight statuses, and adding new flights.
- **Performance Testing and Analysis:**
  The application is lightweight and handles tasks efficiently due to the simplicity of the data structure. Performance is determined by the GUI responsiveness and dictionary lookups, which are optimal for this scale.
- **Optimization and Advanced Features:**
  While the basic implementation works as intended, advanced features like persistent data storage and real-time status updates could be added for scalability.
- **Extensions and Creativity:**
  Extensions could include integration with external APIs for real-time flight updates, a search history feature, or a more advanced GUI with additional user options.

✓ Analysis & Reporting /Answer:

1. The Airport Management System is a GUI-based application created using Python's tkinter library.
2. It allows users to search, view, update, and add flight details efficiently.
3. The system is functional and user-friendly, meeting its basic objectives.
4. It currently uses hardcoded data and lacks persistent storage.
5. Future improvements could include adding a database and real-time flight updates for enhanced functionality.

✓ Lab Activity:

Implementation of a GUI application using Python's tkinter to manage flight data and statuses.

✓ Deliverables:

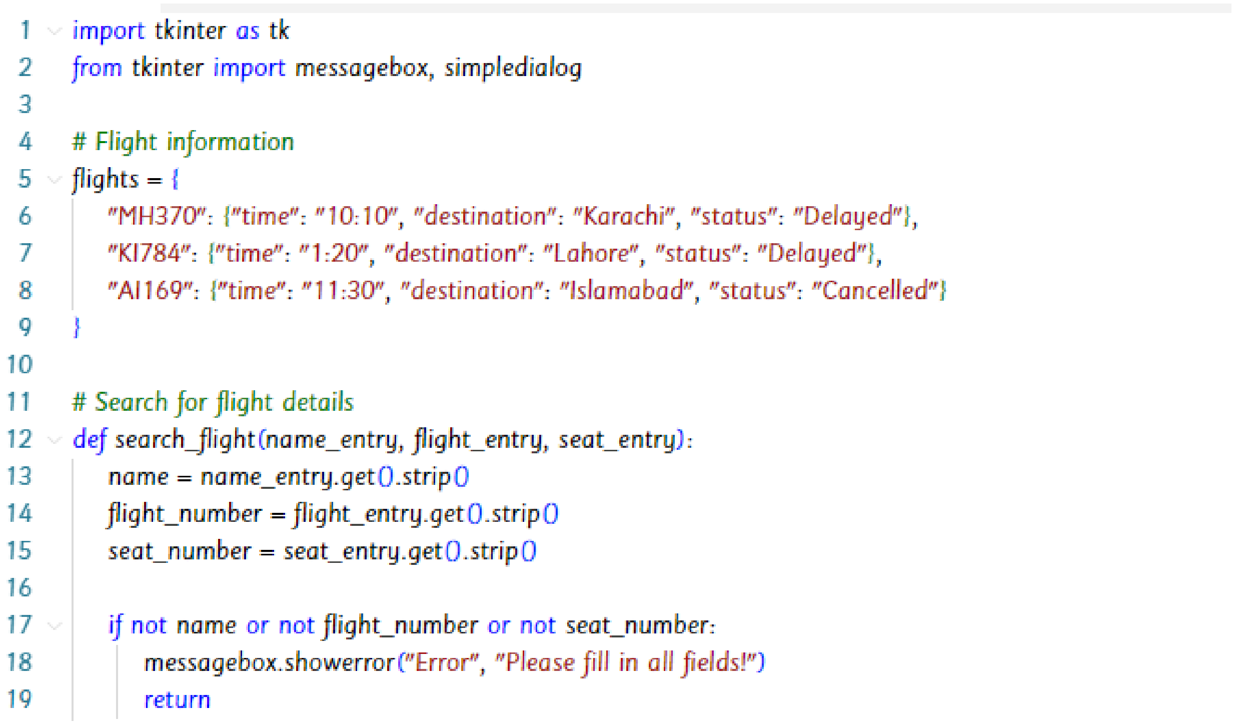
○ Background/Theory:

GUI applications improve user interaction by providing visual elements to manage data. tkinter is a standard library in Python for creating such applications.

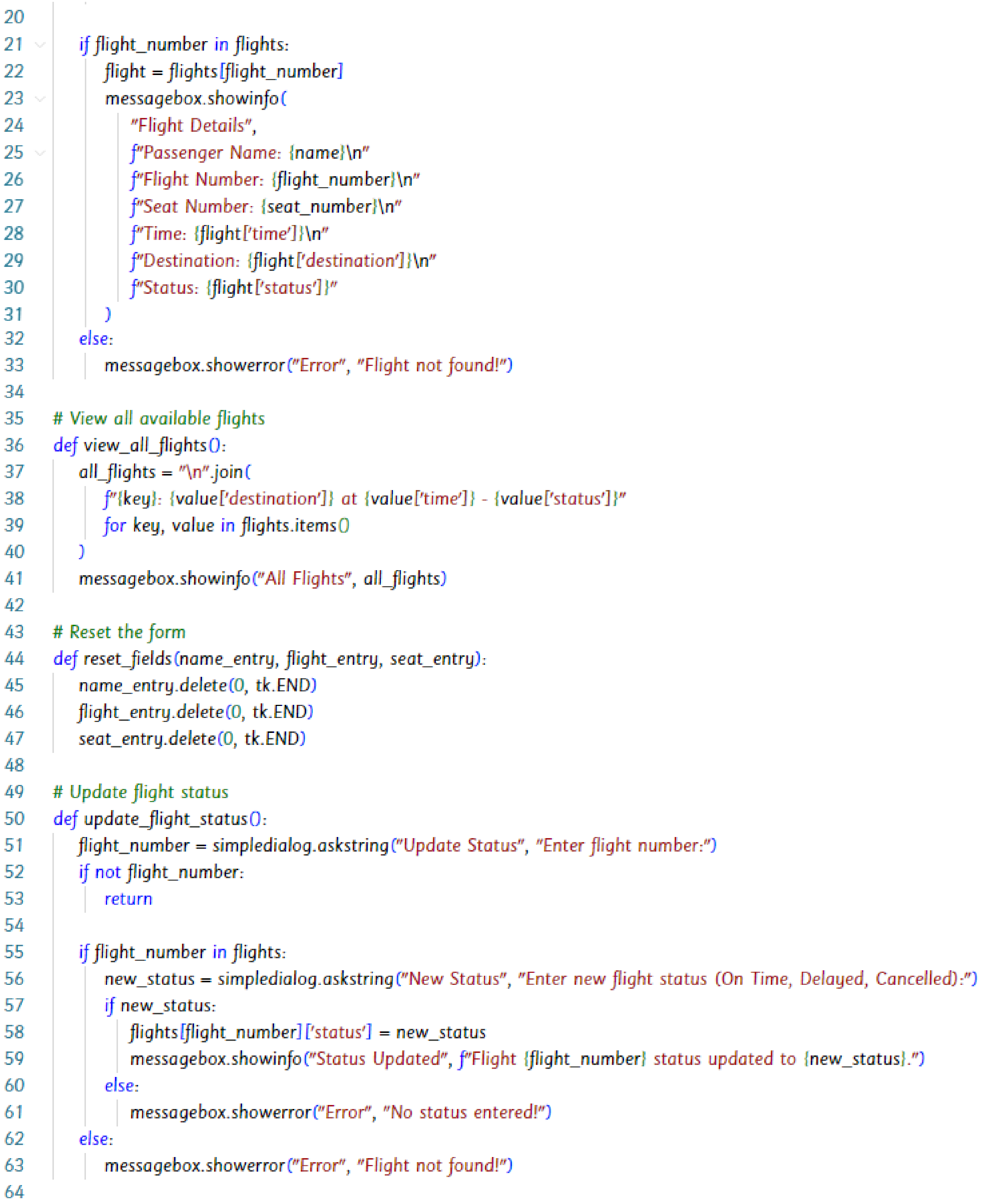
○ Procedure / Methodology:

The project was implemented using an iterative approach. Functions were developed incrementally, starting with basic GUI elements and adding features like search, reset, and update functionalities.

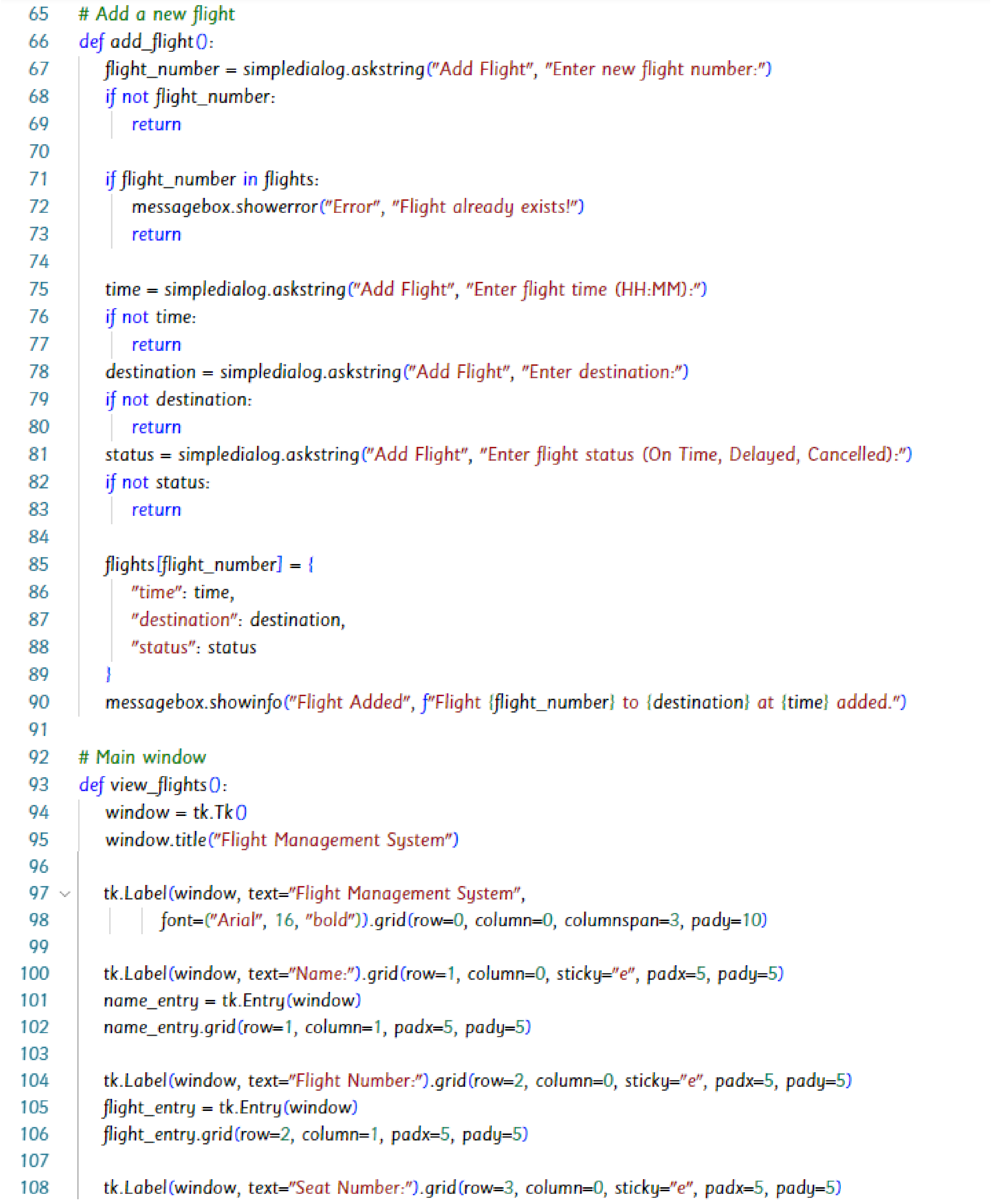
Source Code



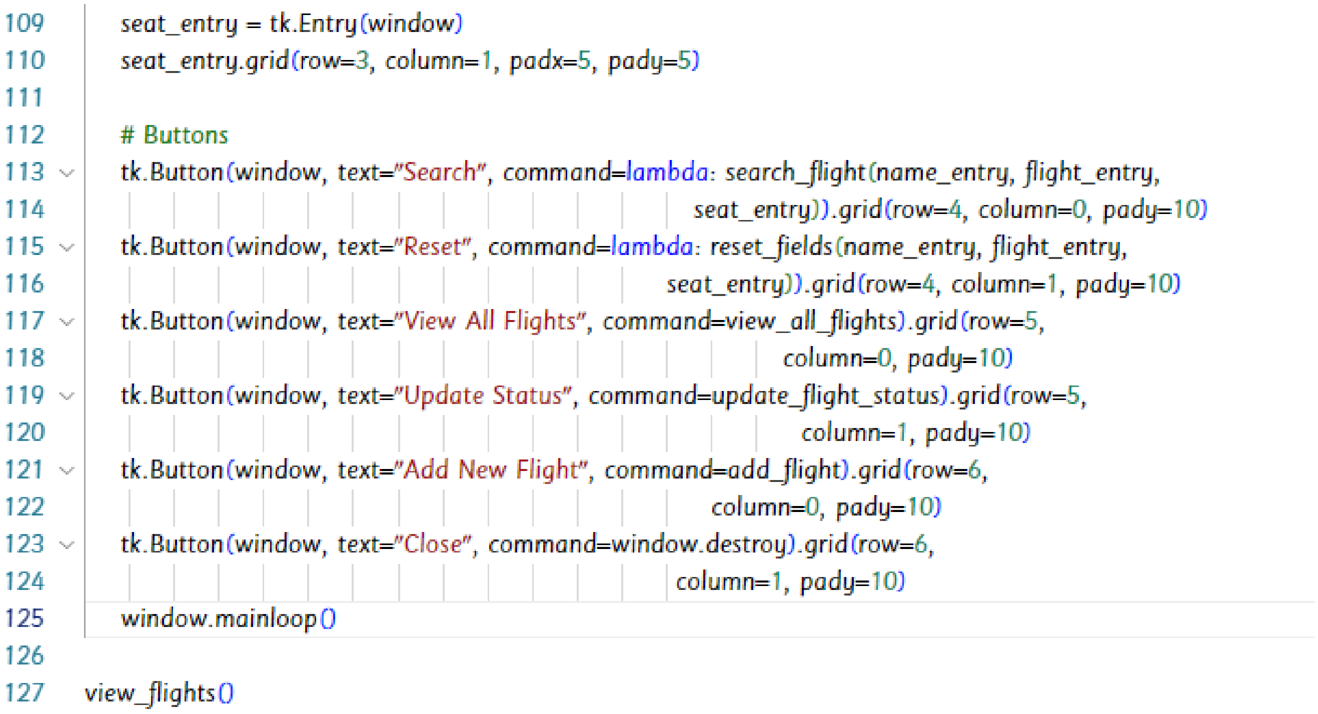
```python
import tkinter as tk
from tkinter import messagebox, simpledialog

# Flight information
flights = {
    "MH370": {"time": "10:10", "destination": "Karachi", "status": "Delayed"},
    "KI784": {"time": "1:20", "destination": "Lahore", "status": "Delayed"},
    "AI169": {"time": "11:30", "destination": "Islamabad", "status": "Cancelled"}
}

# Search for flight details
def search_flight(name_entry, flight_entry, seat_entry):
    name = name_entry.get().strip()
    flight_number = flight_entry.get().strip()
    seat_number = seat_entry.get().strip()

    if not name or not flight_number or not seat_number:
        messagebox.showerror("Error", "Please fill in all fields!")
        return
```



```

    if flight_number in flights:
        flight = flights[flight_number]
        messagebox.showinfo(
            "Flight Details",
            f"Passenger Name: {name}\n"
            f"Flight Number: {flight_number}\n"
            f"Seat Number: {seat_number}\n"
            f"Time: {flight['time']}\n"
            f"Destination: {flight['destination']}\n"
            f"Status: {flight['status']}"
        )
    else:
        messagebox.showerror("Error", "Flight not found!")

# View all available flights
def view_all_flights():
    all_flights = "\n".join(
        f"{key}: {value['destination']} at {value['time']} - {value['status']}"
        for key, value in flights.items()
    )
    messagebox.showinfo("All Flights", all_flights)

# Reset the form
def reset_fields(name_entry, flight_entry, seat_entry):
    name_entry.delete(0, tk.END)
    flight_entry.delete(0, tk.END)
    seat_entry.delete(0, tk.END)

# Update flight status
def update_flight_status():
    flight_number = simpledialog.askstring("Update Status", "Enter flight number:")
    if not flight_number:
        return

    if flight_number in flights:
        new_status = simpledialog.askstring("New Status", "Enter new flight status (On Time, Delayed, Cancelled):")
        if new_status:
            flights[flight_number]['status'] = new_status
            messagebox.showinfo("Status Updated", f"Flight {flight_number} status updated to {new_status}.")
        else:
            messagebox.showerror("Error", "No status entered!")
    else:
        messagebox.showerror("Error", "Flight not found!")

```



```python
# Add a new flight
def add_flight():
    flight_number = simpledialog.askstring("Add Flight", "Enter new flight number:")
    if not flight_number:
        return

    if flight_number in flights:
        messagebox.showerror("Error", "Flight already exists!")
        return

    time = simpledialog.askstring("Add Flight", "Enter flight time (HH:MM):")
    if not time:
        return
    destination = simpledialog.askstring("Add Flight", "Enter destination:")
    if not destination:
        return
    status = simpledialog.askstring("Add Flight", "Enter flight status (On Time, Delayed, Cancelled):")
    if not status:
        return

    flights[flight_number] = {
        "time": time,
        "destination": destination,
        "status": status
    }
    messagebox.showinfo("Flight Added", f"Flight {flight_number} to {destination} at {time} added.")

# Main window
def view_flights():
    window = tk.Tk()
    window.title("Flight Management System")

    tk.Label(window, text="Flight Management System",
            font=("Arial", 16, "bold")).grid(row=0, column=0, columnspan=3, pady=10)

    tk.Label(window, text="Name:").grid(row=1, column=0, sticky="e", padx=5, pady=5)
    name_entry = tk.Entry(window)
    name_entry.grid(row=1, column=1, padx=5, pady=5)

    tk.Label(window, text="Flight Number:").grid(row=2, column=0, sticky="e", padx=5, pady=5)
    flight_entry = tk.Entry(window)
    flight_entry.grid(row=2, column=1, padx=5, pady=5)

    tk.Label(window, text="Seat Number:").grid(row=3, column=0, sticky="e", padx=5, pady=5)
```



```
    seat_entry = tk.Entry(window)
    seat_entry.grid(row=3, column=1, padx=5, pady=5)

    # Buttons
    tk.Button(window, text="Search", command=lambda: search_flight(name_entry, flight_entry,
                                                                  seat_entry)).grid(row=4, column=0, pady=10)
    tk.Button(window, text="Reset", command=lambda: reset_fields(name_entry, flight_entry,
                                                                seat_entry)).grid(row=4, column=1, pady=10)
    tk.Button(window, text="View All Flights", command=view_all_flights).grid(row=5,
                                                                             column=0, pady=10)
    tk.Button(window, text="Update Status", command=update_flight_status).grid(row=5,
                                                                              column=1, pady=10)
    tk.Button(window, text="Add New Flight", command=add_flight).grid(row=6,
                                                                     column=0, pady=10)
    tk.Button(window, text="Close", command=window.destroy).grid(row=6,
                                                                column=1, pady=10)
    window.mainloop()

view_flights()
```

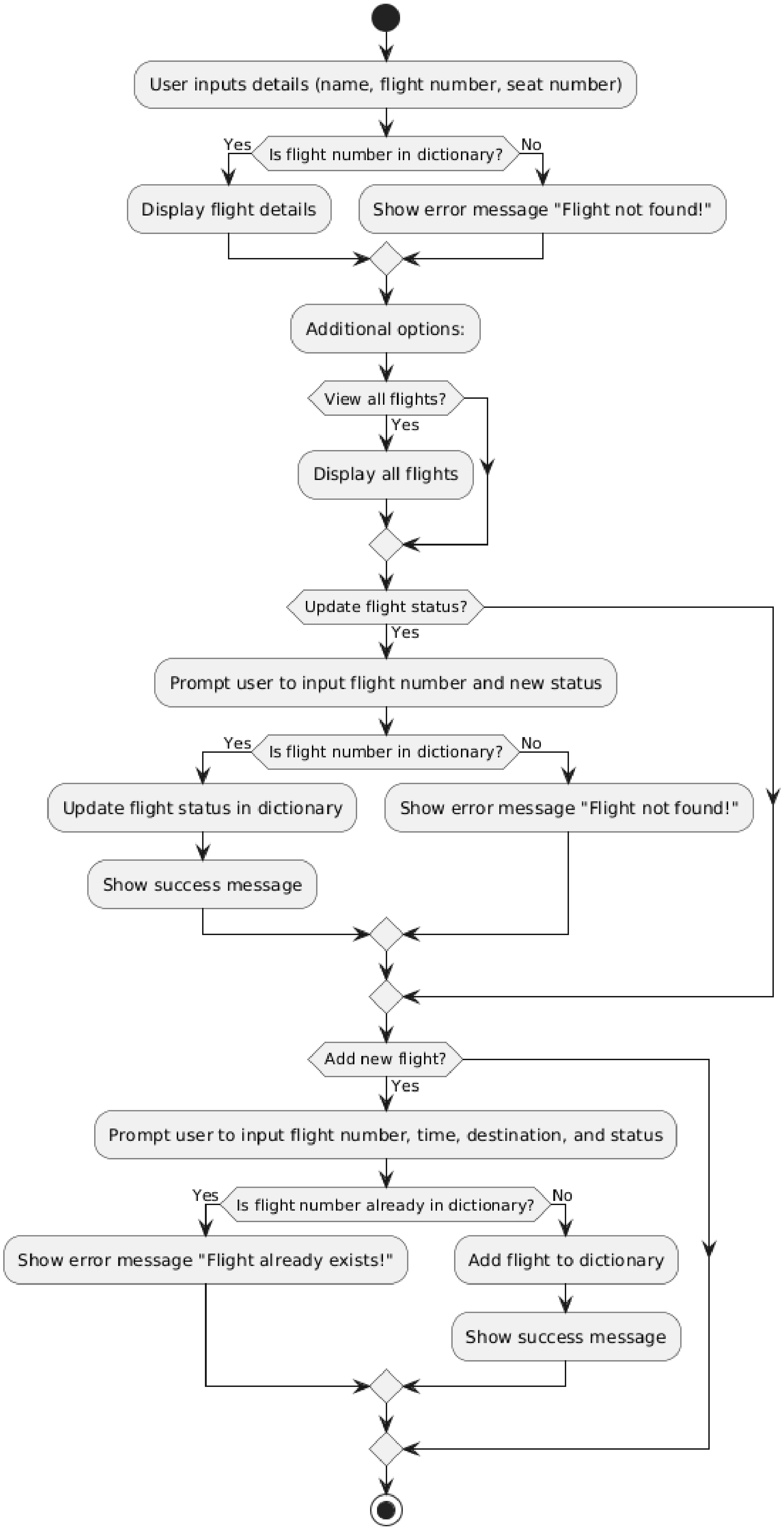
- ## Data Collection (If required):

  Not applicable, as the data is hardcoded in the application for demonstration purposes.

- ## Flowchart:

You can see Flow Chart in Next Page

1. User inputs details (name, flight number, seat number).
2. The system checks the flight number against the dictionary.
3. If found, details are displayed; otherwise, an error message is shown.
4. Additional options allow users to view all flights, update statuses, and add new flights.


User inputs details (name, flight number, seat number)
Is flight number in dictionary?
Yes
No
Display flight details
Show error message "Flight not found!"
Additional options:
View all flights?
Yes
Display all flights
Update flight status?
Yes
Prompt user to input flight number and new status
Is flight number in dictionary?
Yes
No
Update flight status in dictionary
Show error message "Flight not found!"
Show success message
Add new flight?
Yes
Prompt user to input flight number, time, destination, and status
Is flight number already in dictionary?
Yes
No
Show error message "Flight already exists!"
Add flight to dictionary
Show success message

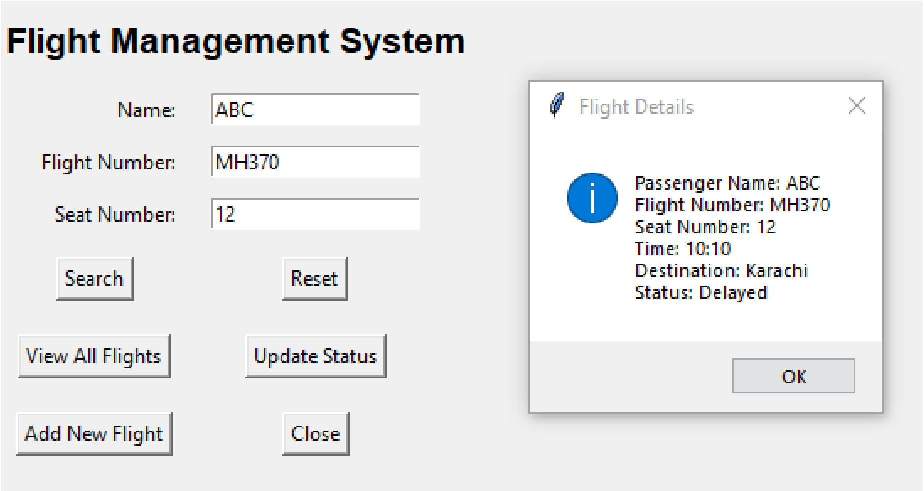
- **Analysis:**

  The system effectively handles the basic requirements for managing flight data. The dictionary lookup ensures fast access to flight information.

- **Results:**

  The application successfully performs the following:

  1. Searches for flight details.



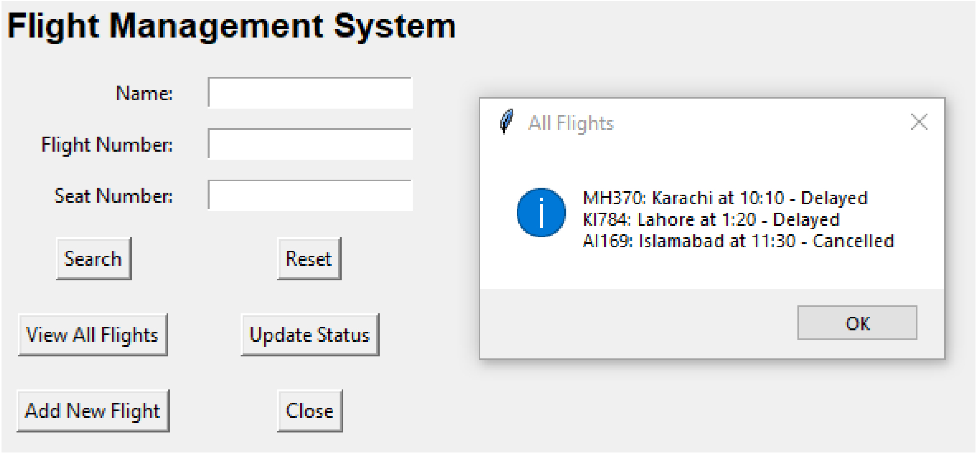
Flight Management System

Name: ABC
Flight Number: MH370
Seat Number: 12

Flight Details

Passenger Name: ABC
Flight Number: MH370
Seat Number: 12
Time: 10:10
Destination: Karachi
Status: Delayed

  2. Displays all available flights.



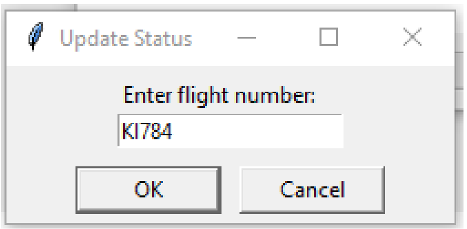
Flight Management System

Name:
Flight Number:
Seat Number:

All Flights

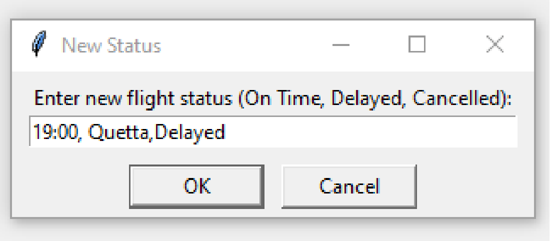
MH370: Karachi at 10:10 - Delayed
KI784: Lahore at 1:20 - Delayed
AI169: Islamabad at 11:30 - Cancelled

  3. Allows users to update flight statuses.



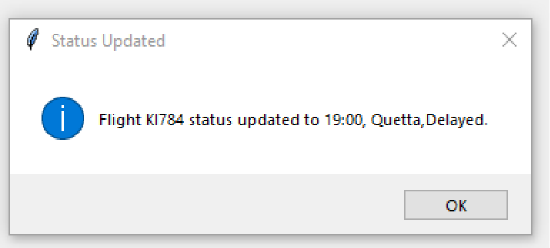
Update Status

Enter flight number:
KI784



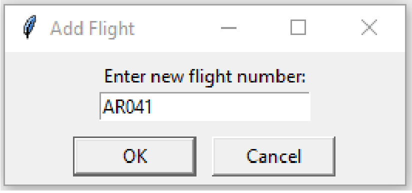
Enter new flight status (On Time, Delayed, Cancelled):

19:00, Quetta,Delayed



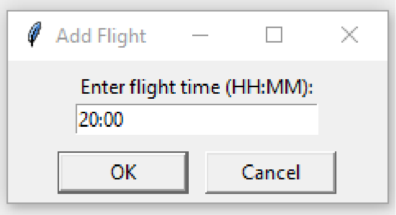
Flight KI784 status updated to 19:00, Quetta,Delayed.

4. Enables users to add new flights to the system.



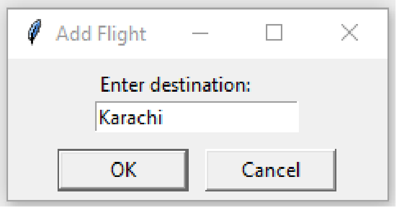
Enter new flight number:

AR041



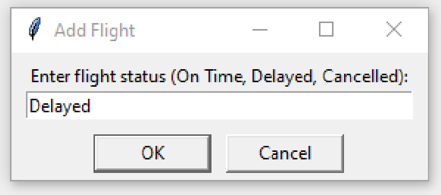
Enter flight time (HH:MM):

20:00



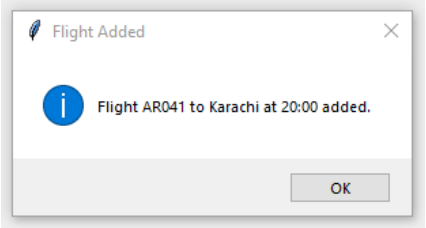
Enter destination:

Karachi



Enter flight status (On Time, Delayed, Cancelled):

Delayed



Flight AR041 to Karachi at 20:00 added.

- **Discussion on Results:**

  The application meets the initial objectives and provides a functional GUI. However, it is limited to hardcoded data and lacks persistent storage, which could enhance usability.

- **Concluding Remarks:**

  The Flight Management System is a functional prototype demonstrating the use of Python and tkinter for GUI-based applications. Future improvements could include data persistence and more advanced user interactions.

- **Reference:**

  Python Official Documentation: https://docs.python.org/3/library/tkinter.html