Flight reservation program

The aims of the flight reservation program are to provide users with a user-friendly interface to perform the following tasks:

**View Available Flights:**

Users can view a list of available flights along with their origins, flight numbers, respective airlines, and the arrival time.

**Book a Flight:**

Users can book a flight by selecting the desired destination city from the available options. The program validates the chosen city and confirms the booking.

**User Interaction:**

The program ensures user interaction by presenting a menu-driven interface, prompting users for their choices, and handling invalid inputs gracefully.

**Exiting the Program:**

Users have the option to exit the program whenever desired.

Overall, the program aims to simulate a flight reservation system by offering a selection of available flights, allowing users to book a flight, and ensuring a smooth user experience by handling user inputs appropriately.

I think the program works fine by meeting the criteria. Regardless, the provided program is designed to simulate a ticket booking system. Here's an analysis of what worked and what didn't, and potential areas for improvement:

**What worked:**

Functionality: The program allows users to view available flights and book them based on their preferences. On selection, it accurately displays flight details and prompts users to book a flight based on available city options.

Input validation: The program processes user inputs efficiently. It prompts the user for valid options and informs them when invalid input is detected.

Structure: The program is logically structured into functions, which improves readability and maintainability.

**What didn't work:**

Lack of clean flow: The program could benefit from clearer flow in certain areas, especially when returning to the main menu after a failed booking attempt.

Error Handling: Although the program provides error messages for invalid inputs, it may require more robust error handling and user guidance for smoother interactions.

Potential improvements:

Improved user experience: Incorporating a more user-friendly interface, such as graphical elements or clearer instructions, could improve the overall user experience.

Improved validation: Adding more detailed error messages or instructions for users when they make incorrect choices can reduce confusion.

Persistence of reservations: Implementing a way to store and display booked flights even after the program ends can improve the user experience.

More detailed information about flights: Consider providing more comprehensive information about available flights, such as departure and arrival times, duration, etc.

Refactoring for Reusability: Refactoring code to make functions more reusable could improve maintainability and scalability.

Improvements in these aspects could contribute to a more user-friendly and scalable flight booking program.

Testing table:

|  |  |  |
| --- | --- | --- |
| **Test Case Description** | **Input** | **Expected Output** |
| Proceed to the menu | **1** | Display the main menu |
| View available flights | **1** | Display available flight options |
| Book a flight without confirming | **2**, **No** | Prompt to go back to the main menu |
| Book a flight with an invalid city name | **2**, **Yes**, **Mumbai** | Error: Invalid city name. Please choose from the available cities. |
| Book a flight with a valid city name | **2**, **Yes**, **London** | Confirmation message: "You have booked a flight to London. Thank you!" |
| View available flights after booking | **1** | Display available flight options with one less available city |
| Exiting the system | **3** | Display "Bye!" message and exit |

Class Flight:

Method \_\_init\_\_(origin, number, airline, arrival):

Set self.origin = origin

Set self.number = number

Set self.airline = airline

Set self.arrival = arrival

Method information():

Print "Flight origin is " + self.origin

Print "Flight number is " + str(self.number)

Print "Airline is " + self.airline

Print "Arrival time is " + str(self.arrival)

Function available\_flights():

Create flight1 instance of Flight with details (Kuwait, 1000, Kuwait Airways, 10.09)

Create flight2 instance of Flight with details (London, 1001, British Airways, 5.00)

Create flight3 instance of Flight with details (Japan, 1002, Japan Airways, 7.30)

Print "1. Kuwait"

Print "2. London"

Print "3. Japan"

Set booking\_info = User input for selecting a flight

If booking\_info is 1:

Display flight1 information using information() method

Else If booking\_info is 2:

Display flight2 information using information() method

Else If booking\_info is 3:

Display flight3 information using information() method

Else:

Print "Invalid choice. Please select a valid flight."

Function book\_flight():

Create available\_cities dictionary with city names and corresponding numbers

Print "1. Kuwait"

Print "2. London"

Print "3. Japan"

Set book\_choice = User input for booking a flight (Yes/No)

If book\_choice is "yes":

While True:

Set user\_city = User input for city name capitalized

If user\_city is in available\_cities:

Print "You have booked a flight to " + user\_city

Print "Thank you!"

Break the loop

Else:

Print "Error: Invalid city name. Please choose from the available cities."

Main Loop:

While True:

Print "Welcome to our website"

Set a = User input (click 1 to proceed)

If a is 1:

Print "Here is our menu"

Print "1. Available flights"

Print "2. Book a flight"

Print "3. Exit"

Try:

Set choice = User input for choice

If choice is 1:

Call available\_flights() function

Else If choice is 2:

Call book\_flight() function

Else If choice is 3:

Print "Bye!"

Break the loop

Else:

Print "Try again. Please enter a valid choice"

Except ValueError:

Print "Invalid input. Please enter a number."

Else:

Break the loop # Exit if user doesn't want to proceed

**Summary:**

A simple flight reservation system with classes for flight and customer is described in the provided code. Seeing available flights, booking flights, and organizing flights are the primary features. Aspects such as the flight number, origin, destination, departure time, and arrival time are defined by the Flight class. User credentials, including the username and password, are stored in the Customer class.

The code structure displays a straightforward flow of interaction where users (Administrators or Customers) can choose from a number of alternatives, such as managing flights, booking a flight, or viewing available flights. Use Case Descriptions, which outline the actions, preconditions, major flows, and postconditions connected to each use case, are used to illustrate each of these interactions.

The pseudo code offers the basic framework for a flight reservation system; however it does not include a thorough implementation of all the features. It outlines the classes, their attributes, and basic methods, aiming to showcase the system's structure and interactions. Further development would involve implementing the functionalities, error handling, and more robust user interactions.