1). Healthcare Industry

Design a Python class ClinicAppointment that manages patient appointments in a clinic. The system should have the following features:

➔ Book Appointment: ● Prompt for patient name, age, mobile number, and preferred doctor.

● Show time slots (10am, 11am, 12pm, 2pm, 3pm).

● Check slot availability and confirm booking.

➔ View/Cancel Appointment:

● Allow patient to view or cancel their appointment using mobile number.

➔ Doctor Availability:

● Maintain a maximum of 3 appointments per time slot per doctor.

➔ Data Persistence:

● Store appointments in memory only (no files/dbs required).

Ans. class ClinicAppointment:

def \_\_init\_\_(self):

# Dictionary structure: {doctor\_name: {time\_slot: [appointments]}}

self.appointments = {}

self.time\_slots = ["10am", "11am", "12pm", "2pm", "3pm"]

def book\_appointment(self):

name = input("Enter patient name: ")

age = input("Enter patient age: ")

mobile = input("Enter mobile number: ")

doctor = input("Enter preferred doctor name: ")

# Initialize doctor's schedule if not already

if doctor not in self.appointments:

self.appointments[doctor] = {slot: [] for slot in self.time\_slots}

print("Available time slots:")

for slot in self.time\_slots:

if len(self.appointments[doctor][slot]) < 3:

print(slot, "(Available)")

else:

print(slot, "(Full)")

slot\_choice = input("Choose a time slot: ")

if slot\_choice not in self.time\_slots:

print("Invalid time slot.")

return

if len(self.appointments[doctor][slot\_choice]) >= 3:

print("Sorry, this slot is full. Choose another slot.")

return

# Add appointment

self.appointments[doctor][slot\_choice].append({

"name": name,

"age": age,

"mobile": mobile

})

print(f"Appointment booked for {name} with Dr. {doctor} at {slot\_choice}.")

def view\_appointment(self):

mobile = input("Enter your mobile number to view appointment: ")

found = False

for doctor, slots in self.appointments.items():

for slot, patients in slots.items():

for patient in patients:

if patient["mobile"] == mobile:

print(f"Appointment Details:\nPatient: {patient['name']}, Age: {patient['age']}, Doctor: {doctor}, Time: {slot}")

found = True

if not found:

print("No appointment found with this mobile number.")

def cancel\_appointment(self):

mobile = input("Enter your mobile number to cancel appointment: ")

found = False

for doctor, slots in self.appointments.items():

for slot, patients in slots.items():

for patient in patients:

if patient["mobile"] == mobile:

patients.remove(patient)

print(f"Appointment for {patient['name']} with Dr. {doctor} at {slot} has been canceled.")

found = True

return

if not found:

print("No appointment found with this mobile number.")

def doctor\_availability(self):

for doctor, slots in self.appointments.items():

print(f"\nDoctor: {doctor}")

for slot, patients in slots.items():

print(f"{slot}: {len(patients)}/3 booked")

# Demo menu to test the class

if \_\_name\_\_ == "\_\_main\_\_":

clinic = ClinicAppointment()

while True:

print("\nClinic Appointment System")

print("1. Book Appointment")

print("2. View Appointment")

print("3. Cancel Appointment")

print("4. Doctor Availability")

print("5. Exit")

choice = input("Enter your choice: ")

if choice == "1":

clinic.book\_appointment()

elif choice == "2":

clinic.view\_appointment()

elif choice == "3":

clinic.cancel\_appointment()

elif choice == "4":

clinic.doctor\_availability()

elif choice == "5":

print("Exiting system.")

break

else:

print("Invalid choice. Try again.")

2). School Management System

Design a Python class SchoolManagement that helps manage student admissions and records. The system should support:

➔ New Admission:

● Collect student name, age, class (1–12), and guardian's mobile number.

● Assign a unique student ID automatically.

● Validate age: must be between 5 and 18.

● Validate mobile number: must be 10 digits.

➔ View Student Details:

● Allow lookup using student ID.

➔ Update Student Info:

● Update mobile number or class.

➔ Remove Student Record:

● Remove a student using their student ID.

➔ Exit System

Ans. class SchoolManagement:

def \_\_init\_\_(self):

self.students = {} # Stores student records with student\_id as key

self.next\_id = 1 # Auto-increment student ID

def new\_admission(self):

name = input("Enter student name: ").strip()

# Validate age

while True:

try:

age = int(input("Enter age (5-18): "))

if 5 <= age <= 18:

break

else:

print("Age must be between 5 and 18.")

except ValueError:

print("Invalid input. Please enter a number.")

# Validate class

while True:

try:

student\_class = int(input("Enter class (1-12): "))

if 1 <= student\_class <= 12:

break

else:

print("Class must be between 1 and 12.")

except ValueError:

print("Invalid input. Please enter a number.")

# Validate mobile number

while True:

mobile = input("Enter guardian's 10-digit mobile number: ").strip()

if mobile.isdigit() and len(mobile) == 10:

break

else:

print("Invalid mobile number. Must be 10 digits.")

student\_id = self.next\_id

self.next\_id += 1

self.students[student\_id] = {

"name": name,

"age": age,

"class": student\_class,

"mobile": mobile

}

print(f"Admission successful! Student ID: {student\_id}")

def view\_student(self):

try:

student\_id = int(input("Enter Student ID to view details: "))

student = self.students.get(student\_id)

if student:

print(f"\nStudent ID: {student\_id}")

print(f"Name: {student['name']}")

print(f"Age: {student['age']}")

print(f"Class: {student['class']}")

print(f"Guardian Mobile: {student['mobile']}\n")

else:

print("Student ID not found.")

except ValueError:

print("Invalid input. Please enter a number.")

def update\_student(self):

try:

student\_id = int(input("Enter Student ID to update: "))

student = self.students.get(student\_id)

if not student:

print("Student ID not found.")

return

print("1. Update Class")

print("2. Update Mobile Number")

choice = input("Enter choice: ")

if choice == "1":

while True:

try:

new\_class = int(input("Enter new class (1-12): "))

if 1 <= new\_class <= 12:

student['class'] = new\_class

print("Class updated successfully.")

break

else:

print("Class must be between 1 and 12.")

except ValueError:

print("Invalid input. Please enter a number.")

elif choice == "2":

while True:

new\_mobile = input("Enter new 10-digit mobile number: ").strip()

if new\_mobile.isdigit() and len(new\_mobile) == 10:

student['mobile'] = new\_mobile

print("Mobile number updated successfully.")

break

else:

print("Invalid mobile number. Must be 10 digits.")

else:

print("Invalid choice.")

except ValueError:

print("Invalid input. Please enter a number.")

def remove\_student(self):

try:

student\_id = int(input("Enter Student ID to remove: "))

if student\_id in self.students:

del self.students[student\_id]

print(f"Student record with ID {student\_id} removed successfully.")

else:

print("Student ID not found.")

except ValueError:

print("Invalid input. Please enter a number.")

# Demo menu to test the class

if \_\_name\_\_ == "\_\_main\_\_":

school = SchoolManagement()

while True:

print("\nSchool Management System")

print("1. New Admission")

print("2. View Student Details")

print("3. Update Student Info")

print("4. Remove Student Record")

print("5. Exit System")

choice = input("Enter your choice: ")

if choice == "1":

school.new\_admission()

elif choice == "2":

school.view\_student()

elif choice == "3":

school.update\_student()

elif choice == "4":

school.remove\_student()

elif choice == "5":

print("Exiting system.")

break

else:

print("Invalid choice. Try again.")

3). Transport Reservation System (Bus Ticketing)

Design a Python class BusReservation that simulates a basic bus ticket booking system. Features should include:

➔ Show Available Routes:

● Predefined city routes with fixed prices.

● Example: "Mumbai to Pune - ₹500", "Delhi to Jaipur - ₹600", etc.

➔ Book Ticket:

● Enter passenger name, age, mobile, and route.

● Assign seat number (max 40 per bus per route).

● Generate a unique ticket ID.

➔ View Ticket:

● Lookup using ticket ID.

➔ Cancel Ticket:

● Cancel the ticket if it exists.

➔ Exit

Ans. class BusReservation:

def \_\_init\_\_(self):

# Predefined routes with prices

self.routes = {

"Mumbai to Pune": 500,

"Delhi to Jaipur": 600,

"Bangalore to Mysore": 400,

"Chennai to Coimbatore": 450

}

self.tickets = {} # Stores ticket info with ticket\_id as key

self.route\_seats = {route: [] for route in self.routes} # Track booked seats per route

self.next\_ticket\_id = 1

def show\_routes(self):

print("\nAvailable Routes:")

for route, price in self.routes.items():

print(f"{route} - ₹{price}")

def book\_ticket(self):

name = input("Enter passenger name: ").strip()

while True:

try:

age = int(input("Enter passenger age: "))

break

except ValueError:

print("Invalid input. Please enter a number.")

while True:

mobile = input("Enter 10-digit mobile number: ").strip()

if mobile.isdigit() and len(mobile) == 10:

break

else:

print("Invalid mobile number. Must be 10 digits.")

self.show\_routes()

route = input("Choose your route exactly as shown: ").strip()

if route not in self.routes:

print("Invalid route selected.")

return

# Check seat availability

if len(self.route\_seats[route]) >= 40:

print("Sorry, no seats available on this route.")

return

seat\_number = len(self.route\_seats[route]) + 1

ticket\_id = self.next\_ticket\_id

self.next\_ticket\_id += 1

ticket\_info = {

"name": name,

"age": age,

"mobile": mobile,

"route": route,

"seat\_number": seat\_number,

"price": self.routes[route]

}

self.tickets[ticket\_id] = ticket\_info

self.route\_seats[route].append(seat\_number)

print(f"Ticket booked successfully! Ticket ID: {ticket\_id}, Seat Number: {seat\_number}")

def view\_ticket(self):

try:

ticket\_id = int(input("Enter Ticket ID to view: "))

ticket = self.tickets.get(ticket\_id)

if ticket:

print("\nTicket Details:")

print(f"Ticket ID: {ticket\_id}")

print(f"Passenger Name: {ticket['name']}")

print(f"Age: {ticket['age']}")

print(f"Mobile: {ticket['mobile']}")

print(f"Route: {ticket['route']}")

print(f"Seat Number: {ticket['seat\_number']}")

print(f"Price: ₹{ticket['price']}\n")

else:

print("Ticket ID not found.")

except ValueError:

print("Invalid input. Please enter a number.")

def cancel\_ticket(self):

try:

ticket\_id = int(input("Enter Ticket ID to cancel: "))

ticket = self.tickets.pop(ticket\_id, None)

if ticket:

self.route\_seats[ticket['route']].remove(ticket['seat\_number'])

print(f"Ticket ID {ticket\_id} canceled successfully.")

else:

print("Ticket ID not found.")

except ValueError:

print("Invalid input. Please enter a number.")

# Demo menu to test the class

if \_\_name\_\_ == "\_\_main\_\_":

bus\_system = BusReservation()

while True:

print("\nBus Reservation System")

print("1. Show Available Routes")

print("2. Book Ticket")

print("3. View Ticket")

print("4. Cancel Ticket")

print("5. Exit")

choice = input("Enter your choice: ")

if choice == "1":

bus\_system.show\_routes()

elif choice == "2":

bus\_system.book\_ticket()

elif choice == "3":

bus\_system.view\_ticket()

elif choice == "4":

bus\_system.cancel\_ticket()

elif choice == "5":

print("Exiting system.")

break

else:

print("Invalid choice. Try again.")