OUP OF INSTITUTIONS

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

AUTONOMOUS INSTITUTE UNDER VISVESVARAYA TECHNOLOGICAL UNIVERSITY
JNANA SANGAMA, BELAGAVI 590018

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Seminar Report on

"Sports clinic appointment scheduler"

For the course: Python

Name:- Nisarga P	**Usn no:-22BI24EC402-T**
Hema M	**22BI24EC405-T**
Priyanka bai	**22BI24EC403-T**
Mounika	**22BI24EC408-T**

BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

NACC Accredited institution*
(Recognised by Govt.of karnataka,approved by AICTE,NewDelhi & Affiliated to Visvesvaraya Technological University, Belgavi)
"Jnana Gangotri" Campus,No.873/2,Ballari-Hospet Road,Allipur,Ballari-583104
Karnataka,India.
Ph: 08392-237100/23719, Fax:08392-237197

2023-2024

INTRODUCTION

SPORTS CLINIC APPOINTMENT SCHEDULER

A sports clinic appointment scheduler is a system that allows patients to book appointments for sports-related activities at a clinic:

- Appointment scheduling: A vital process that helps clinics manage patient appointments efficiently.
- Dynamic scheduling: A scheduling method that can improve clinic performance by allocating capacity based on patient requests and capacity fluctuations.
- Online appointment scheduling: A system that allows patients to book appointments online.
- Payment options: A system that allows customers to pay for appointments at the time of booking or by sending an invoice afterwards.
- Time management: A system that helps sports facilities manage their time.

Problem Statement: Sports Clinic Appointment Scheduler

Background

In a sports clinic, managing patient appointments efficiently is crucial for providing timely care. This program is designed to help clinic staff schedule, manage, and keep track of patient appointments.

Objectives

Develop a Python program that allows clinic staff to perform the following tasks:

Add Patients: Staff can enter patient details, including name, phone number, and email address. The system should prevent duplicate entries for the same patient.

Schedule Appointments: Staff can schedule appointments for existing patients by specifying the date and time. The system should confirm the appointment and handle cases where the patient does not exist.

Cancel Appointments: Staff can cancel an existing appointment by the patient's name. The system should confirm the cancellation or inform if no appointment is found.

Reschedule Appointments: Staff can reschedule an existing appointment by updating the date and time. The system should confirm the changes or inform if no appointment is found.

List Appointments: Staff can view all scheduled appointments, displaying the patient's name, date, and time of the appointment.

Detailed analysis of code components

1.Class Definitions

a. Patient Class

Purpose: Represents a patient with relevant details.

Attributes:

name: The name of the patient.
phone: The contact number of the patient.
email: The email address of the patient.

Constructor (_init_):

Initializes the patient’s attributes with provided values.

b. Appointment Class

Purpose: Represents an appointment for a patient.

Attributes:

patient: An instance of the Patient class associated with the appointment.
date: The date of the appointment (in the format YYYY-MM-DD).
time: The time of the appointment (in the format HH).

Constructor (_init_):

Initializes the appointment's attributes with the patient and specified date and time.

c. Scheduler Class

Purpose: Manages the operations related to patients and appointments.

Attributes:

patients: A dictionary to store patients using their names as keys.
appointments: A dictionary to store appointments using patient names as keys.

2. Methods in Scheduler Class

a. add_patient
Parameters: name, phone, email

Functionality:

Checks if the patient name already exists in the patients dictionary.

If not, creates a new Patient object and adds it to the dictionary, providing feedback to the user.

b. schedule_appointment

Parameters: name, date, time

Functionality:

Looks up the patient by name in the patients dictionary.

If the patient exists, creates an Appointment object and adds it to the appointments dictionary, providing confirmation. If not found, it gives an error message.

c. cancel_appointment

Parameters: name

Functionality:
Checks if an appointment exists for the given patient name in the appointments dictionary.

If found, deletes the appointment and provides confirmation. If not found,

it gives an error message.

d. reschedule_appointment

Parameters: name, new_date, new_time

Functionality:

Checks if an appointment exists for the patient name.

If found, updates the appointment's date and time and provides confirmation. If not found, it gives an error message.

e. list_appointments

Functionality:

Iterates through the appointments dictionary and prints details of all scheduled appointments. If no appointments exist, it informs the user.

3. User Interface (UI) Loop

A while loop that continuously presents a menu to the user until they choose to exit.

Menu Options:

Adds a patient.
Schedules an appointment.
Cancels an appointment.
Reschedules an appointment.
Lists all appointments.
Exits the program.

User Input Handling:

Based on user input, the corresponding method in the Scheduler class is called.

Each action is followed by user feedback indicating success or failure.

4. Error Handling and User Feedback

The program includes simple feedback mechanisms for:

Duplicate patients.
Non-existent patients when scheduling or canceling.
Confirmation messages for successful operations.
Although basic, these provide essential usability.

5. Potential Enhancements

Input Validation: Implement checks to ensure the date and time formats are correct and valid.

Duplicate Appointment Checks: Prevent scheduling multiple appointments for the same patient at the same time.

Data Persistence: Use a database or file storage to save patients and appointments between program runs.

Exception Handling: Incorporate try-except blocks to manage unexpected errors gracefully.

ALGORITHM

Algorithm for Appointment Scheduler

- ❖ Initialize Scheduler

Create an instance of the Scheduler class.

Initialize empty dictionaries for patients and appointments.

Display Menu Options

Print options for user actions:

Add Patient

Schedule Appointment

Cancel Appointment

Reschedule Appointment

List Appointments

Exit

User Input

Prompt the user to select an option from the menu.

Handle User Choices

Choice 1: Add Patient

Prompt for patient name, phone, and email.

Validate:

Check if name already exists in patients.

Validate phone format.

Validate email format.

If valid, create a new Patient object and add to patients dictionary.

Print confirmation.

Choice 2: Schedule Appointment

Prompt for patient name, date, and time.

Check if the patient exists in patients.

Check for appointment conflicts:

If there's an existing appointment for the same name, date, and time, inform the user.

If valid, create a new Appointment object and add to appointments dictionary.

Print confirmation.

Choice 3: Cancel Appointment

Prompt for patient name.

Check if an appointment exists for that name in appointments.

If found, delete the appointment and print confirmation.

If not found, inform the user.

Choice 4: Reschedule Appointment

Prompt for patient name, new date, and new time.

Check if an appointment exists for that name.

If found, update the date and time of the existing appointment.

Print confirmation.

Choice 5: List Appointments

If there are no appointments, print a message.

If there are appointments, iterate through the appointments dictionary and print each appointment's details.

Choice 6: Exit

Terminate the program.

Repeat
Continue displaying the menu until the user selects the exit option.

Program:

```
Class Patient:
  def _init_(self, name, phone, email):
        self.name = name
        self.Phone = phone
        self.email = email

class Appointment:
    def _init_(self, patient, date, time):
        self.patient = patient
        self.date = date
        self.time = time
```

```
class Scheduler:
    def _init_(self):
        self.patients = {}
        self.appointments = {}

    def add_patient(self, name, phone, email):
        patient = Patient(name, phone, email)
        self.patients[name] = patient

    def schedule_appointment(self, name, date, time):
        patient = self.patients.get(name)
        if patient:
            appointment = Appointment(patient, date, time)
            self.appointments[name] = appointment
            print(f"Appointment scheduled for {name} on {date} at {time}")
        else:
            print("Patient not found")

    def cancel_appointment(self, name):
        if name in self.appointments:
            del self.appointments[name]
            print(f"Appointment cancelled for {name}")
        else:
            print("Appointment not found")

    def reschedule_appointment(self, name, new_date, new_time):
        if name in self.appointments:
            appointment = self.appointments[name]
            appointment.date = new_date
            appointment.time = new_time
            print(f"Appointment rescheduled for {name} on {new_date} at
{new_time}")
        else:
            print("Appointment not found")


# Usage
scheduler = Scheduler()
```

```
while True:
    print("\nSports Clinic Appointment Scheduler")
    print("1. Add Patient")
    print("2. Schedule Appointment")
    print("3. Cancel Appointment")
    print("4. Reschedule Appointment")
    print("5. Exit")

    choice = input("Choose an option: ")

    if choice == "1":
        name = input("Enter patient name: ")
        phone = input("Enter patient phone: ")
        email = input("Enter patient email: ")
        scheduler.add_patient(name, phone, email)
    elif choice == "2":
        name = input("Enter patient name: ")
        date = input("Enter appointment date (YYYY-MM-DD): ")
        time = input("Enter appointment time (HH:MM): ")
        scheduler.schedule_appointment(name, date, time)
    elif choice == "3":
        name = input("Enter patient name: ")
        scheduler.cancel_appointment(name)
    elif choice == "4":
        name = input("Enter patient name: ")
      new_date = input("Enter new appointment date (YYYY-MM-DD): ")
        new_time = input("Enter new appointment time (HH:MM): ")
        scheduler.reschedule_appointment(name, new_date, new_time)
    elif choice == "5":
        break
    else:
        print("Invalid option")
```

Output

```
Sports Clinic Appointment Scheduler
1. Add Patient
2. Schedule Appointment
3. Cancel Appointment
4. Reschedule Appointment
5. Exit
Choose an option: 1
Enter patient name: nisarga
Enter patient phone: 9476767512
Enter patient email: nisarga@gmail.com

Sports Clinic Appointment Scheduler
1. Add Patient
2. Schedule Appointment
3. Cancel Appointment
4. Reschedule Appointment
5. Exit
Choose an option: 2
Enter patient name: nisarga
Enter appointment date (YYYY-MM-DD): 2024-10-27
Enter appointment time (HH:MM): 3:45
Appointment scheduled for nisarga on 2024-10-27 at 3:45
```

```
Sports Clinic Appointment Scheduler
1. Add Patient
2. Schedule Appointment
3. Cancel Appointment
4. Reschedule Appointment
5. Exit
Choose an option: 4
Enter patient name: nisarga
Enter new appointment date (YYYY-MM-DD): 2024-09-10
Enter new appointment time (HH:MM): 2:35
Appointment rescheduled for nisarga on 2024-09-10 at 2:35
```

Conclusion:

The provided code implements a straightforward appointment scheduling system for a sports clinic. It effectively manages patient information and appointments through object-oriented programming. Key functionalities include adding patients, scheduling, canceling, and rescheduling appointments.

Key Takeaways:

Modularity: The use of classes (Patient, Appointment, and Scheduler) promotes clean organization and separation of concerns, making the code easier to maintain and extend.

User Interaction: The loop-driven menu facilitates user interaction, allowing for a seamless experience when managing appointments.

Input Validation: Implementing validation for email and phone formats improves data integrity and user feedback, ensuring valid entries.

Future Enhancements: Potential improvements could include data persistence, advanced error handling, and user-friendly features like viewing all appointments or searching for specific patients.

Overall, this code serves as a solid foundation for a more comprehensive appointment management system, suitable for further development based on specific user needs or clinic requirements.