

**PYTHON PROJECT**

|  |  |
| --- | --- |
| **Name of the Student** | Prachi Agarwalla |
| **Course** | DSA |
| **Program** | BTech CSE |
| **Sap ID** | 500126504 |
| **Batch No.** | 41 |

**TOPIC- INFIRMARY MANAGEMENT SYSTEM**

**Submitted to - Deepak Kumar Sir**

**Domain :-**

Domain name – Infirmary Management System

Domain Details:

It is a comprehensive program designed to simplify healthcare administration and optimize health care within medical facilities. This program presents a menu prompting a user to choose between admin mode and user mode. In admin mode, the user is prompted to enter a password, and upon successful authentication, they gain access to functionalities such as managing patients, doctors, and medicines. For example, in the patient management section, the admin can add, edit, display, or delete patient records stored in a text file. Similarly, functionalities for managing doctors and medicines are also provided. The user mode, on the other hand, allows users to log in or sign up. After logging in, the user can see his/her profile or can order medicines. Overall, this system serves as a rudimentary framework for managing various aspects of an infirmary, including patient records, doctor information, and medicine inventory.

Top of Form

**Reason:**

Managing a medical facility involves handling various aspects, including patient records, doctor schedules, medicine inventory and billing. Designing an IMS project provides an opportunity to tackle complex problems and develop robust solutions.

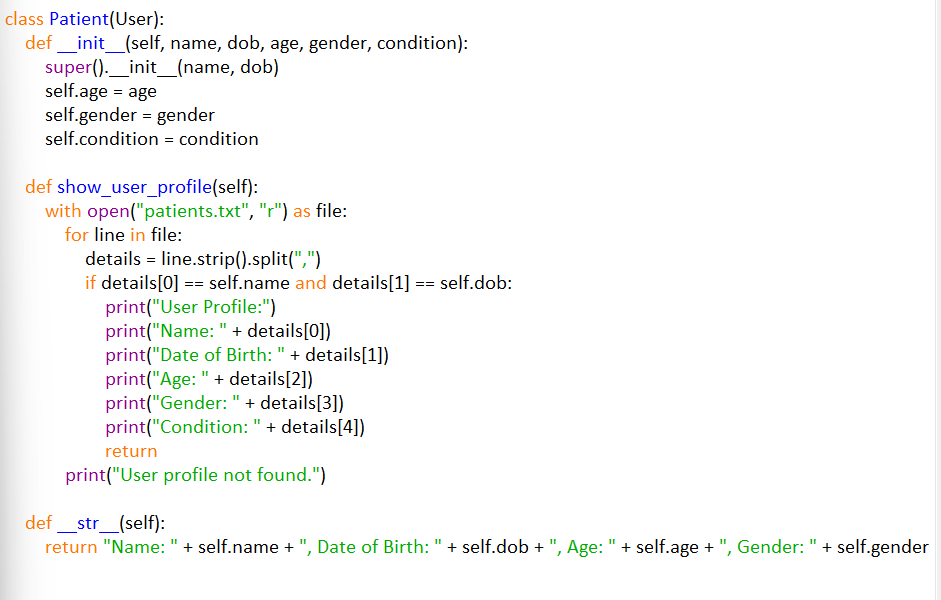
**OVERVIEW:**

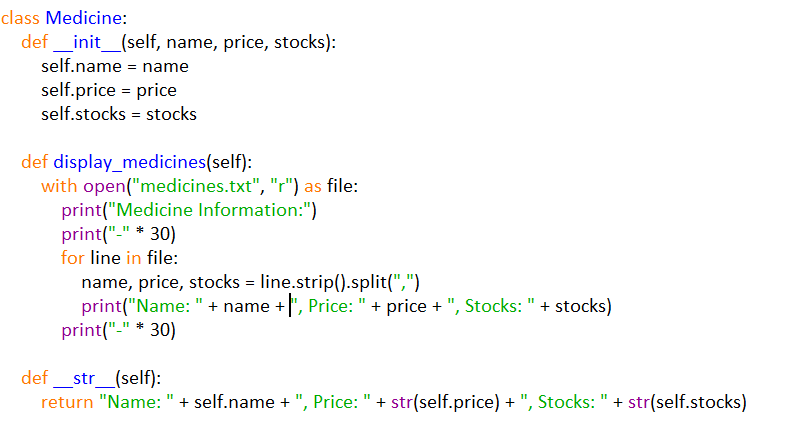
The program is structured into following classes:

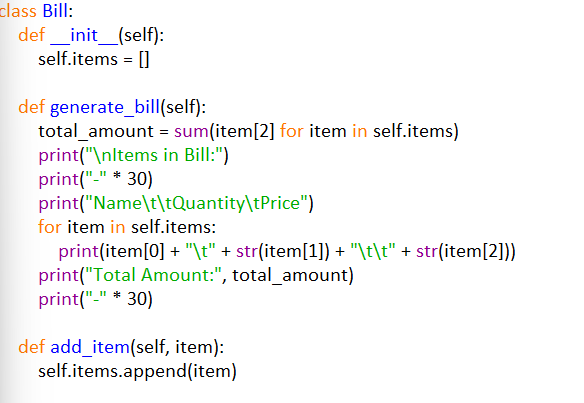
1. **User:**
   * **\_\_init\_\_(self, name, dob)**: Initializes a user with a name and date of birth.
   * **login(self)**: Allows a user to log in by providing their name and date of birth. It reads the login information from a file named "patients.txt".
   * **signup(self)**: Allows a new user to sign up by providing their information such as name, date of birth, age, gender, and condition. This information is appended to "patients.txt".



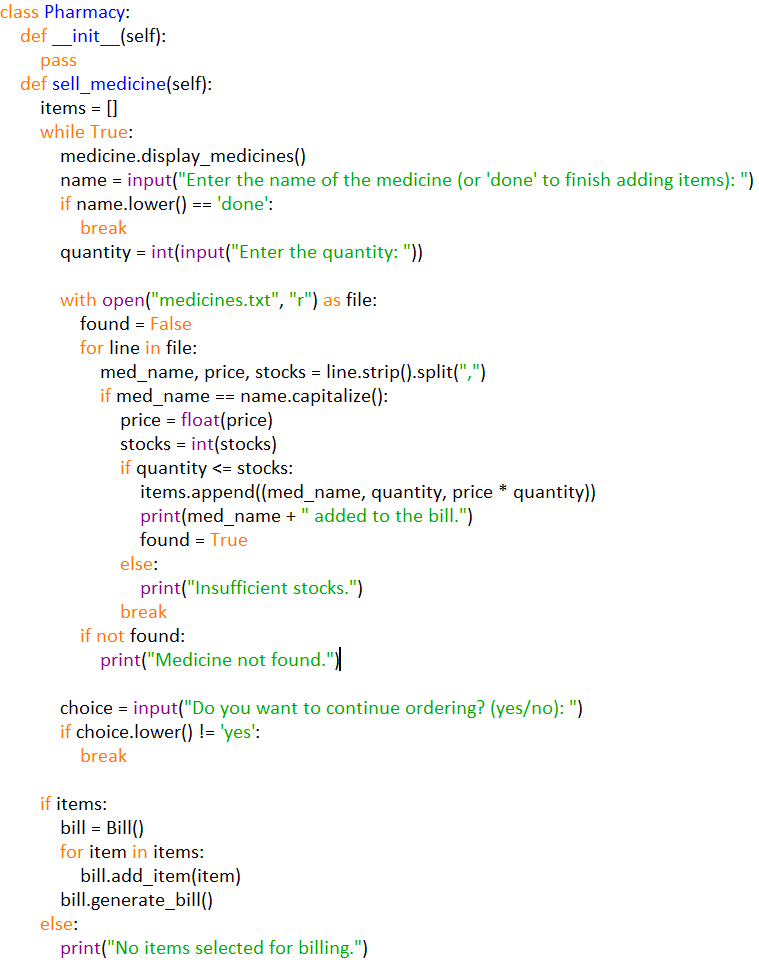
1. **Patient (Inherits from User):**
   * **\_\_init\_\_(self, name, dob, age, gender, condition)**: Initializes a patient with a name, date of birth, age, gender, and condition.
   * **show\_user\_profile(self)**: Displays the profile of the logged-in patient.
   * **\_\_str\_\_(self)**: Returns a formatted string of patient information.



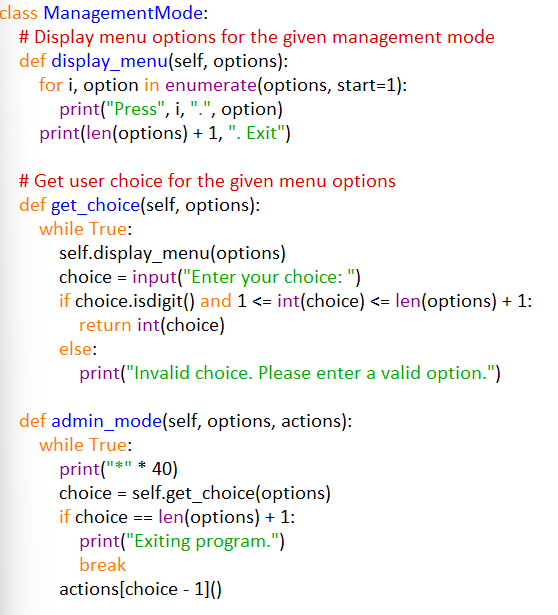
1. **Medicine:**
   * **\_\_init\_\_(self, name, price, stocks)**: Initializes a medicine with a name, price, and available stocks.
   * **display\_medicines(self)**: Displays information about all available medicines.
   * **\_\_str\_\_(self)**: Returns a formatted string of medicine information.
   * 
2. **Bill:**
   * **\_\_init\_\_(self)**: Initializes an empty bill.
   * **generate\_bill(self)**: Generates and displays the bill with the items, quantities, and total amount.
   * **add\_item(self, item)**: Adds an item to the bill.



1. **Pharmacy:**
   * **\_\_init\_\_(self)**: Initializes the Pharmacy object.
   * **sell\_medicine(self)**: Allows the user to purchase medicines, adding them to the bill.

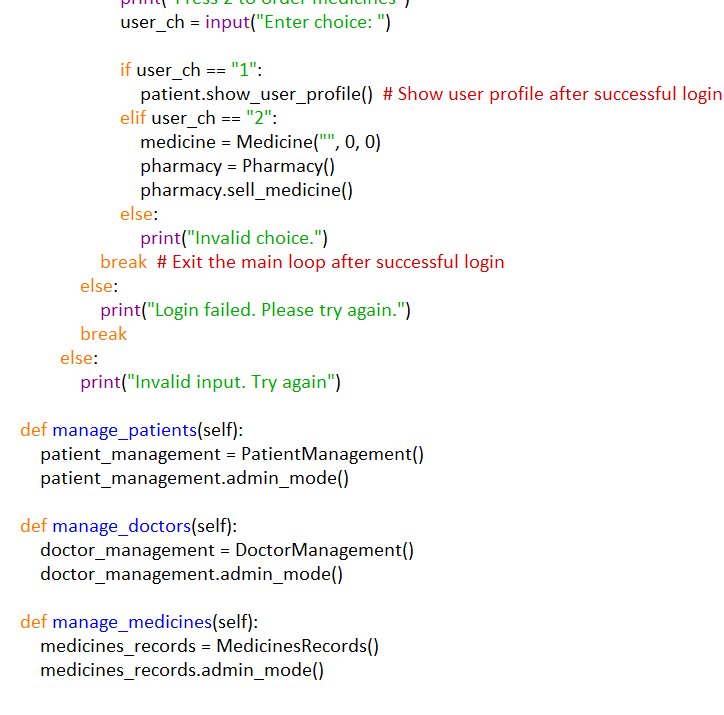


1. **ManagementMode:**
   * **display\_menu(self, options)**: Displays the menu for the given mode.
   * **get\_choice(self, options)**: Gets the user's choice from the menu.
   * **admin\_mode(self, options, actions)**: Provides the admin mode functionality. It takes options and corresponding actions to execute.



1. **Infirmary (Inherits from ManagementMode):**
   * **\_\_init\_\_(self)**: Initializes the Infirmary object.
   * **start(self)**: The main function that runs the infirmary. It gives the user the option to choose admin mode or user mode.
   * **manage\_patients(self)**: Redirects to the PatientManagement class.
   * **manage\_doctors(self)**: Redirects to the DoctorManagement class.
   * **manage\_medicines(self)**: Redirects to the MedicinesRecords class.





1. **PatientManagement (Inherits from ManagementMode):**
   * **\_\_init\_\_(self)**: Initializes the PatientManagement object.
   * **display\_patients(self)**: Displays information about all patients.
   * **add\_patient(self)**: Allows admins to add a new patient.
   * **edit\_patient(self)**: Allows admins to edit patient information.
   * **delete\_patient(self)**: Allows admins to delete a patient from the records.
   * **admin\_mode(self)**: Executes admin mode functionality specific to patient management.

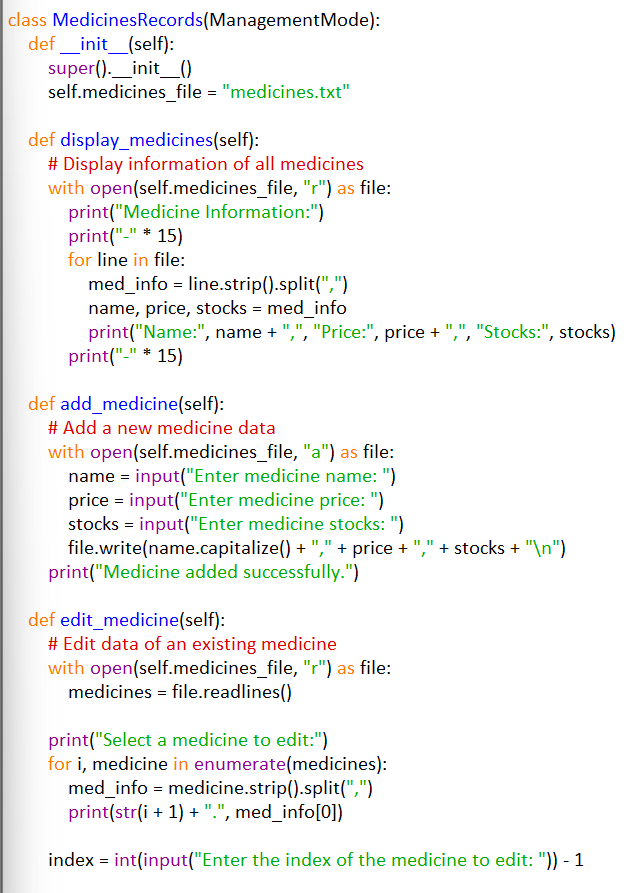


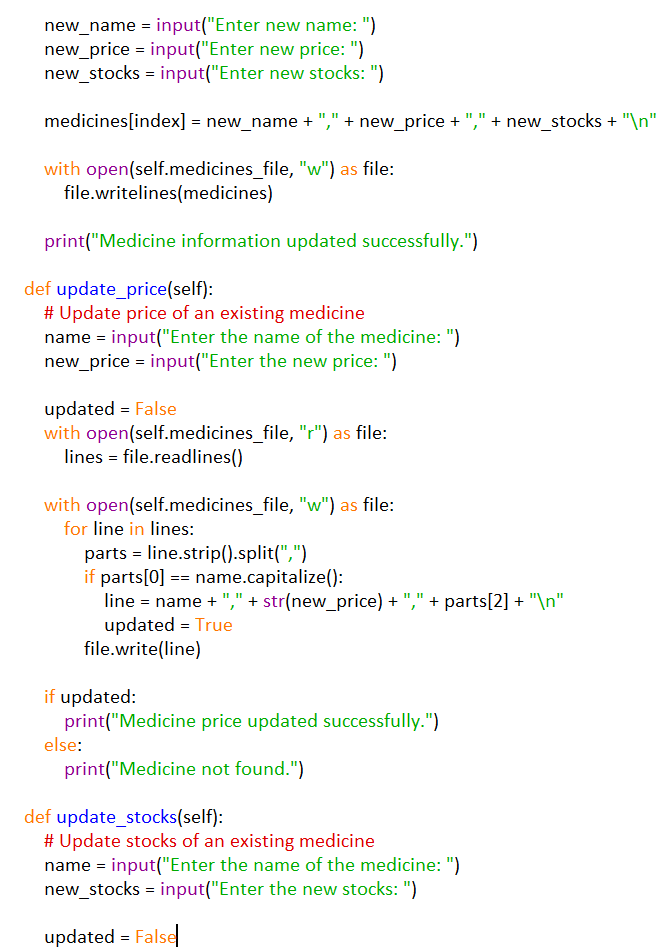


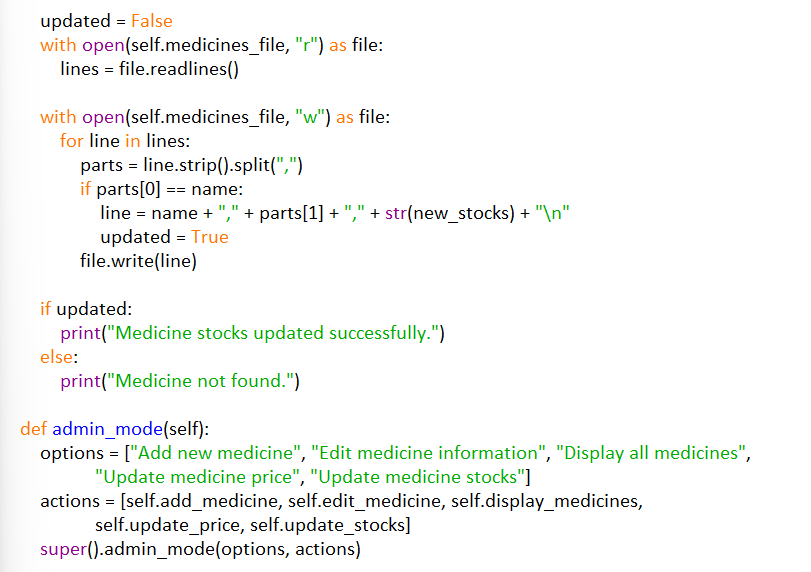
1. **DoctorManagement (Inherits from ManagementMode):**
   * **\_\_init\_\_(self)**: Initializes the DoctorManagement object.
   * **display\_doctors(self)**: Displays information about all doctors.
   * **add\_doctor(self)**: Allows admins to add a new doctor.
   * **admin\_mode(self)**: Executes admin mode functionality specific to doctor management.



1. **MedicinesRecords (Inherits from ManagementMode):**
   * **\_\_init\_\_(self)**: Initializes the MedicinesRecords object.
   * **display\_medicines(self)**: Displays information about all medicines.
   * **add\_medicine(self)**: Allows admins to add a new medicine.
   * **edit\_medicine(self)**: Allows admins to edit medicine information.
   * **update\_price(self)**: Allows admins to update the price of a medicine.
   * **update\_stocks(self)**: Allows admins to update the stocks of a medicine.
   * **admin\_mode(self)**: Executes admin mode functionality specific to managing medicine records.

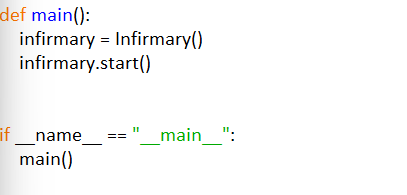






1. **main():**

**main()**: Main function that initializes the Infirmary object and starts the infirmary.



**Main Code :**

**class User:**

**def \_\_init\_\_(self, name, dob):**

**self.name = name**

**self.dob = dob**

**def login(self):**

**name = input("Enter your name: ")**

**dob = input("Enter your password (DD/MM/YYYY): ")**

**login\_successful = False #Initialize login status flag**

**with open("patients.txt", "r") as file:**

**for line in file:**

**details = line.strip().split(",")**

**if details[0] == name and details[1] == dob:**

**print("Login successful!")**

**login\_successful = True**

**break #Exit loop once login is successful**

**if not login\_successful:**

**print("Invalid credentials. Please try again")**

**return name, dob, login\_successful**

**def signup(self):**

**with open("patients.txt", "a") as file:**

**name = input("Enter your name: ")**

**dob = input("Enter your date of birth (DDMMYYYY): ")**

**age = input("Enter age: ")**

**gender = input("Enter gender(M/F/O): ")**

**condition = input("Enter condition: ")**

**patient = name + "," + dob + "," + age + "," + gender + "," + condition + "\n"**

**file.write(str(patient) + "\n")**

**print("Account created successfully.")**

**class Patient(User):**

**def \_\_init\_\_(self, name, dob, age, gender, condition):**

**super().\_\_init\_\_(name, dob)**

**self.age = age**

**self.gender = gender**

**self.condition = condition**

**def show\_user\_profile(self):**

**with open("patients.txt", "r") as file:**

**for line in file:**

**details = line.strip().split(",")**

**if details[0] == self.name and details[1] == self.dob:**

**print("User Profile:")**

**print("Name: " + details[0])**

**print("Date of Birth: " + details[1])**

**print("Age: " + details[2])**

**print("Gender: " + details[3])**

**print("Condition: " + details[4])**

**return**

**print("User profile not found.")**

**def \_\_str\_\_(self):**

**return "Name: " + self.name + ", Date of Birth: " + self.dob + ", Age: " + self.age + ", Gender: " + self.gender + ", Condition: " + self.condition**

**class Medicine:**

**def \_\_init\_\_(self, name, price, stocks):**

**self.name = name**

**self.price = price**

**self.stocks = stocks**

**def display\_medicines(self):**

**with open("medicines.txt", "r") as file:**

**print("Medicine Information:")**

**print("-" \* 30)**

**for line in file:**

**name, price, stocks = line.strip().split(",")**

**print("Name: " + name + ", Price: " + price + ", Stocks: " + stocks)**

**print("-" \* 30)**

**def \_\_str\_\_(self):**

**return "Name: " + self.name + ", Price: " + str(self.price) + ", Stocks: " + str(self.stocks)**

**class Bill:**

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

**self.items = []**

**def generate\_bill(self):**

**total\_amount = sum(item[2] for item in self.items)**

**print("\nItems in Bill:")**

**print("-" \* 30)**

**print("Name\t\tQuantity\tPrice")**

**for item in self.items:**

**print(item[0] + "\t" + str(item[1]) + "\t\t" + str(item[2]))**

**print("Total Amount:", total\_amount)**

**print("-" \* 30)**

**def add\_item(self, item):**

**self.items.append(item)**

**class Pharmacy:**

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

**pass**

**def sell\_medicine(self):**

**items = []**

**while True:**

**medicine.display\_medicines()**

**name = input("Enter the name of the medicine (or 'done' to finish adding items): ")**

**if name.lower() == 'done':**

**break**

**quantity = int(input("Enter the quantity: "))**

**with open("medicines.txt", "r") as file:**

**found = False**

**for line in file:**

**med\_name, price, stocks = line.strip().split(",")**

**if med\_name == name.capitalize():**

**price = float(price)**

**stocks = int(stocks)**

**if quantity <= stocks:**

**items.append((med\_name, quantity, price \* quantity))**

**print(med\_name + " added to the bill.")**

**found = True**

**else:**

**print("Insufficient stocks.")**

**break**

**if not found:**

**print("Medicine not found.")**

**choice = input("Do you want to continue ordering? (yes/no): ")**

**if choice.lower() != 'yes':**

**break**

**if items:**

**bill = Bill()**

**for item in items:**

**bill.add\_item(item)**

**bill.generate\_bill()**

**else:**

**print("No items selected for billing.")**

**class ManagementMode:**

**# Display menu options for the given management mode**

**def display\_menu(self, options):**

**for i, option in enumerate(options, start=1):**

**print("Press", i, ".", option)**

**print(len(options) + 1, ". Exit")**

**# Get user choice for the given menu options**

**def get\_choice(self, options):**

**while True:**

**self.display\_menu(options)**

**choice = input("Enter your choice: ")**

**if choice.isdigit() and 1 <= int(choice) <= len(options) + 1:**

**return int(choice)**

**else:**

**print("Invalid choice. Please enter a valid option.")**

**def admin\_mode(self, options, actions):**

**while True:**

**print("\*" \* 40)**

**choice = self.get\_choice(options)**

**if choice == len(options) + 1:**

**print("Exiting program.")**

**break**

**actions[choice - 1]()**

**class Infirmary(ManagementMode):**

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

**super().\_\_init\_\_()**

**def start(self):**

**while True:**

**print("\*" \* 55)**

**print(" " \* 10 + "WELCOME TO OUR INFIRMARY" + " " \* 10)**

**print("\*" \* 55)**

**print("Press 1 for admin mode and 2 for user mode")**

**mode = input("Enter your mode: ")**

**if mode == "1":**

**password = input("Please enter your password: ")**

**if password == "care@108":**

**print("\*" \* 51)**

**print(" " \* 15 + "Welcome to admin mode" + " " \* 15)**

**print("\*" \* 51)**

**options = ["Manage patients", "Manage doctors", "Manage medicines"]**

**actions = [self.manage\_patients, self.manage\_doctors, self.manage\_medicines]**

**super().admin\_mode(options, actions)**

**break**

**else:**

**print("Incorrect password. Please try again.")**

**elif mode == "2":**

**print("\*" \* 50)**

**print(" " \* 15 + "Welcome to user mode" + " " \* 15)**

**print("\*" \* 50)**

**user = User("", "")**

**name, dob, login\_successful = user.login() # Receive name, dob, and login status from login**

**if login\_successful: # Check if login was successful**

**patient = Patient(name, dob, "", "", "")**

**while True:**

**print("\nHow can we help you?")**

**print("Press 1 to show user profile")**

**print("Press 2 to order medicines")**

**user\_ch = input("Enter choice: ")**

**if user\_ch == "1":**

**patient.show\_user\_profile() # Show user profile after successful login**

**elif user\_ch == "2":**

**medicine = Medicine("", 0, 0)**

**pharmacy = Pharmacy()**

**pharmacy.sell\_medicine()**

**else:**

**print("Invalid choice.")**

**break # Exit the main loop after successful login**

**else:**

**print("Login failed. Please try again.")**

**break**

**else:**

**print("Invalid input. Try again")**

**def manage\_patients(self):**

**patient\_management = PatientManagement()**

**patient\_management.admin\_mode()**

**def manage\_doctors(self):**

**doctor\_management = DoctorManagement()**

**doctor\_management.admin\_mode()**

**def manage\_medicines(self):**

**medicines\_records = MedicinesRecords()**

**medicines\_records.admin\_mode()**

**class PatientManagement(ManagementMode):**

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

**super().\_\_init\_\_()**

**self.patients\_file = "patients.txt"**

**def display\_patients(self):**

**# Display information of all patients**

**with open(self.patients\_file, "r") as file:**

**print("Patient Information:")**

**print("-" \* 30)**

**for line in file:**

**details = line.strip().split(",")**

**name, dob, age, gender, condition = details**

**print("Name:", name + ",", "DOB:", dob + ",", "Age:", age + ",", "Gender:", gender + ",", "Condition:", condition)**

**print("-" \* 30)**

**def add\_patient(self):**

**# Add a new patient to the records**

**with open(self.patients\_file, "a") as file:**

**name = input("Enter name: ")**

**dob = input("Enter date of birth in the form (DDMMYYYY): ")**

**age = input("Enter age: ")**

**gender = input("Enter gender: ")**

**condition = input("Enter condition: ")**

**patient = name + "," + dob + "," + age + "," + gender + "," + condition + "\n"**

**file.write(patient)**

**print("Patient added successfully.")**

**def edit\_patient(self):**

**# Edit information of the existing patient**

**with open(self.patients\_file, "r") as file:**

**patients = file.readlines()**

**print("Select a patient to edit:")**

**for i, patient in enumerate(patients):**

**print(str(i + 1) + ".", patient.strip())**

**index = int(input("Enter the index of the patient to edit: ")) - 1**

**new\_name = input("Enter new name: ")**

**new\_dob = input("Enter new dob(DDMMYYYY): ")**

**new\_age = input("Enter new age: ")**

**new\_gender = input("Enter new gender: ")**

**new\_condition = input("Enter new condition: ")**

**patients[index] = new\_name + "," + new\_dob + "," + new\_age + "," + new\_gender + "," + new\_condition + "\n"**

**with open(self.patients\_file, "w") as file:**

**file.writelines(patients)**

**print("Patient information updated successfully.")**

**def delete\_patient(self):**

**# Delete a patient details from the records**

**with open(self.patients\_file, "r") as file:**

**patients = file.readlines()**

**print("Select a patient to delete:")**

**for i, patient in enumerate(patients):**

**print(str(i + 1) + ".", patient.strip())**

**index = int(input("Enter the index of the patient to delete: ")) - 1**

**deleted\_patient = patients.pop(index)**

**with open(self.patients\_file, "w") as file:**

**file.writelines(patients)**

**print(deleted\_patient.strip() + " has been deleted.")**

**def admin\_mode(self):**

**options = ["Add new patient", "Edit patient information", "Display all patients", "Delete a patient"]**

**actions = [self.add\_patient, self.edit\_patient, self.display\_patients, self.delete\_patient]**

**super().admin\_mode(options, actions)**

**class DoctorManagement(ManagementMode):**

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

**super().\_\_init\_\_()**

**self.doctors\_file = "doctors.txt"**

**def display\_doctors(self):**

**# Display information of all existing doctors**

**with open(self.doctors\_file, "r") as file:**

**print("Doctor Information:")**

**print("-" \* 30)**

**for line in file:**

**details = line.strip().split(",")**

**name, age, spl = details**

**print("Name:", name + ",", "Age:", age + ",", "Specialisation:", spl)**

**print("-" \* 30)**

**def add\_doctor(self):**

**# Add a new doctor in the records**

**with open(self.doctors\_file, "a") as file:**

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

**age = input("Enter age: ")**

**spl = input("Enter specialisation: ")**

**file.write(name + "," + age + "," + spl + "\n")**

**print("Doctor added successfully.")**

**def admin\_mode(self):**

**options = ["Add new doctor", "Display all doctors"]**

**actions = [self.add\_doctor, self.display\_doctors]**

**super().admin\_mode(options, actions)**

**class MedicinesRecords(ManagementMode):**

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

**super().\_\_init\_\_()**

**self.medicines\_file = "medicines.txt"**

**def display\_medicines(self):**

**# Display information of all medicines**

**with open(self.medicines\_file, "r") as file:**

**print("Medicine Information:")**

**print("-" \* 15)**

**for line in file:**

**med\_info = line.strip().split(",")**

**name, price, stocks = med\_info**

**print("Name:", name + ",", "Price:", price + ",", "Stocks:", stocks)**

**print("-" \* 15)**

**def add\_medicine(self):**

**# Add a new medicine data**

**with open(self.medicines\_file, "a") as file:**

**name = input("Enter medicine name: ")**

**price = input("Enter medicine price: ")**

**stocks = input("Enter medicine stocks: ")**

**file.write(name.capitalize() + "," + price + "," + stocks + "\n")**

**print("Medicine added successfully.")**

**def edit\_medicine(self):**

**# Edit data of an existing medicine**

**with open(self.medicines\_file, "r") as file:**

**medicines = file.readlines()**

**print("Select a medicine to edit:")**

**for i, medicine in enumerate(medicines):**

**med\_info = medicine.strip().split(",")**

**print(str(i + 1) + ".", med\_info[0])**

**index = int(input("Enter the index of the medicine to edit: ")) - 1**

**new\_name = input("Enter new name: ")**

**new\_price = input("Enter new price: ")**

**new\_stocks = input("Enter new stocks: ")**

**medicines[index] = new\_name + "," + new\_price + "," + new\_stocks + "\n"**

**with open(self.medicines\_file, "w") as file:**

**file.writelines(medicines)**

**print("Medicine information updated successfully.")**

**def update\_price(self):**

**# Update price of an existing medicine**

**name = input("Enter the name of the medicine: ")**

**new\_price = input("Enter the new price: ")**

**updated = False**

**with open(self.medicines\_file, "r") as file:**

**lines = file.readlines()**

**with open(self.medicines\_file, "w") as file:**

**for line in lines:**

**parts = line.strip().split(",")**

**if parts[0] == name.capitalize():**

**line = name + "," + str(new\_price) + "," + parts[2] + "\n"**

**updated = True**

**file.write(line)**

**if updated:**

**print("Medicine price updated successfully.")**

**else:**

**print("Medicine not found.")**

**def update\_stocks(self):**

**# Update stocks of an existing medicine**

**name = input("Enter the name of the medicine: ")**

**new\_stocks = input("Enter the new stocks: ")**

**updated = False**

**with open(self.medicines\_file, "r") as file:**

**lines = file.readlines()**

**with open(self.medicines\_file, "w") as file:**

**for line in lines:**

**parts = line.strip().split(",")**

**if parts[0] == name:**

**line = name + "," + parts[1] + "," + str(new\_stocks) + "\n"**

**updated = True**

**file.write(line)**

**if updated:**

**print("Medicine stocks updated successfully.")**

**else:**

**print("Medicine not found.")**

**def admin\_mode(self):**

**options = ["Add new medicine", "Edit medicine information", "Display all medicines",**

**"Update medicine price", "Update medicine stocks"]**

**actions = [self.add\_medicine, self.edit\_medicine, self.display\_medicines,**

**self.update\_price, self.update\_stocks]**

**super().admin\_mode(options, actions)**

**def main():**

**infirmary = Infirmary()**

**infirmary.start()**

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

**main()**

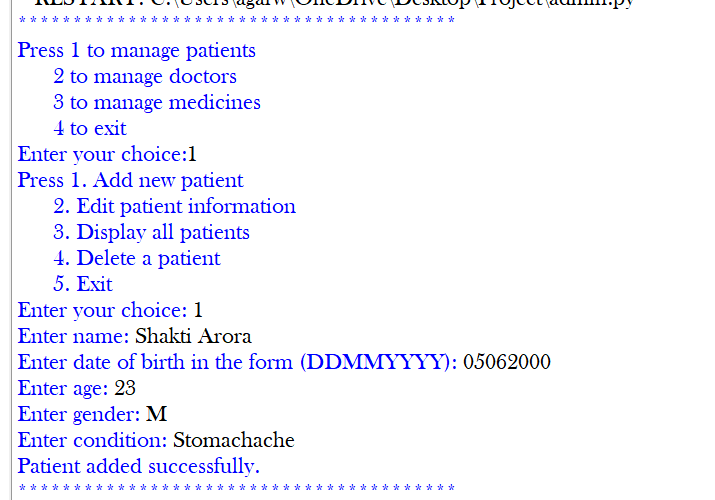
**OUTPUTS:**

**Choose mode**

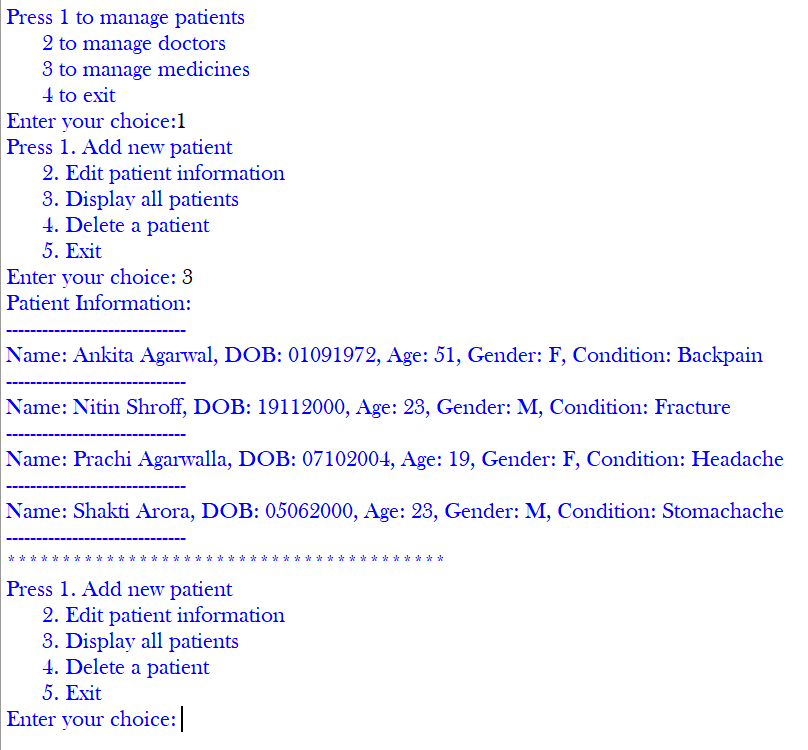
****

**1.Admin Mode**

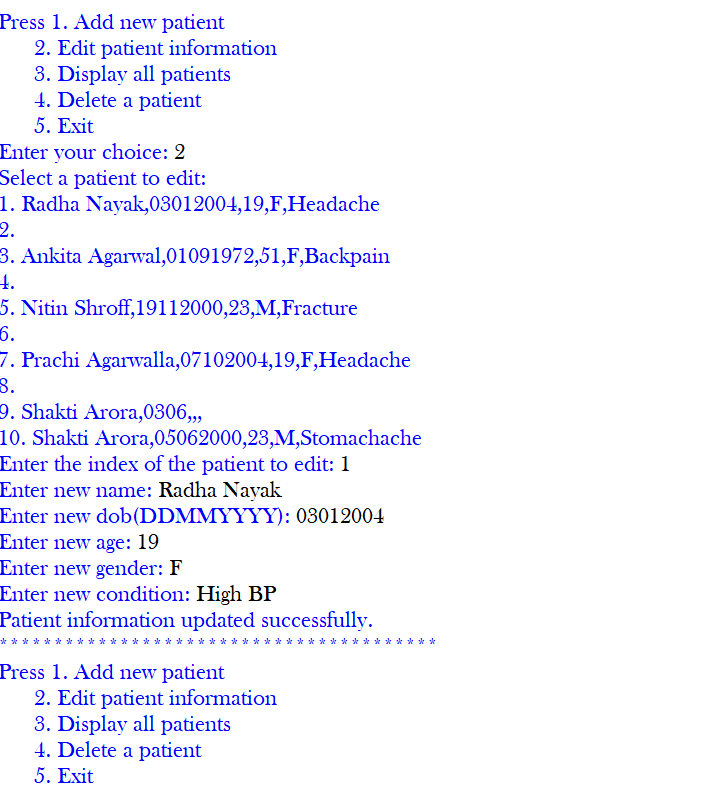
**Add new patient**

****

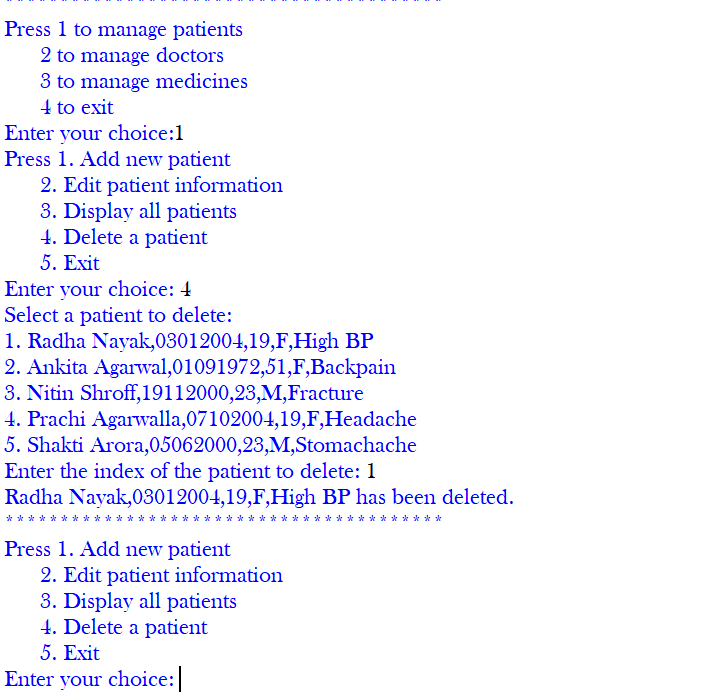
**Display patients**

****

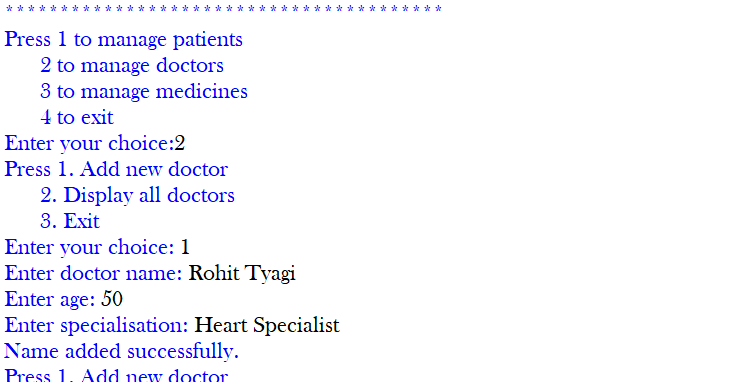
**Edit patient data**

****

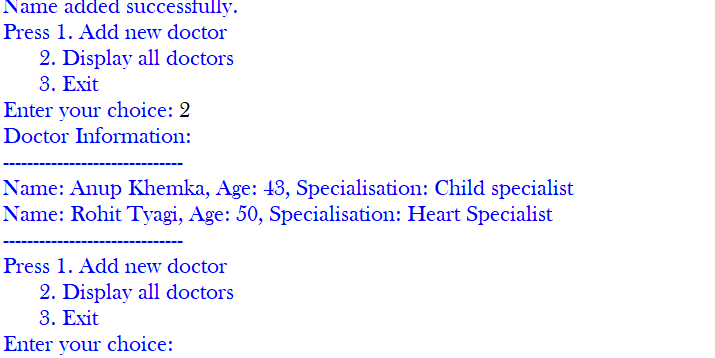
**Delete patient data**

****

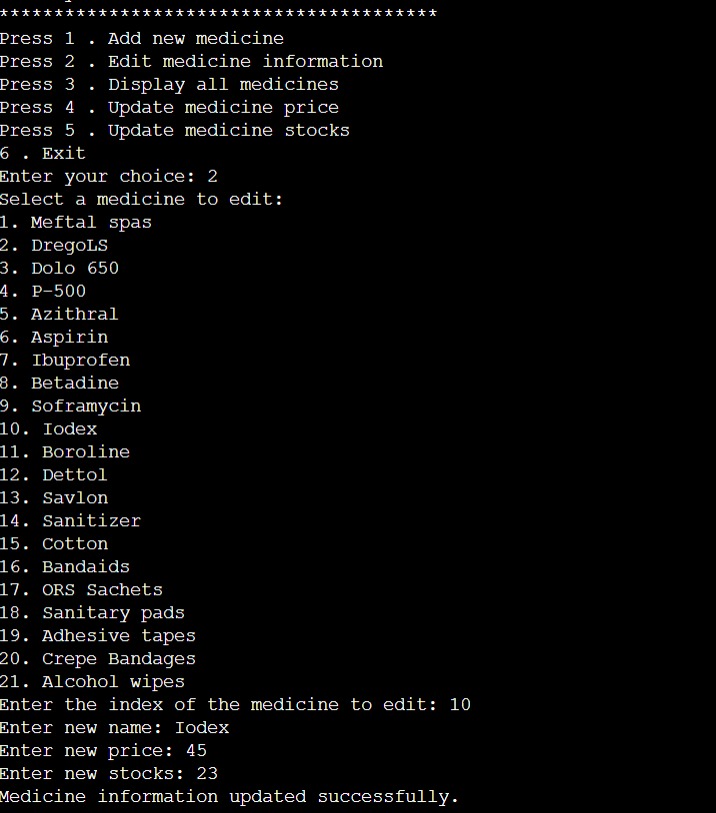
**Add doctor**

****

**Display doctors data**

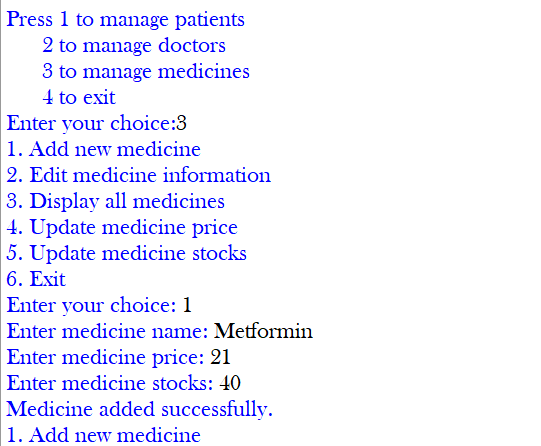
****

**Edit medicine information**

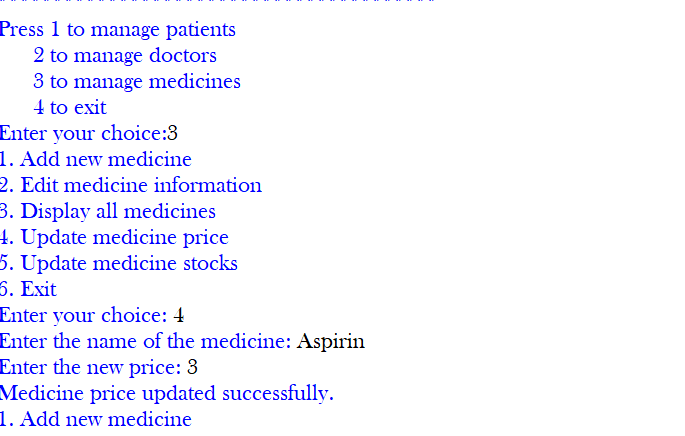


**Display medicines**

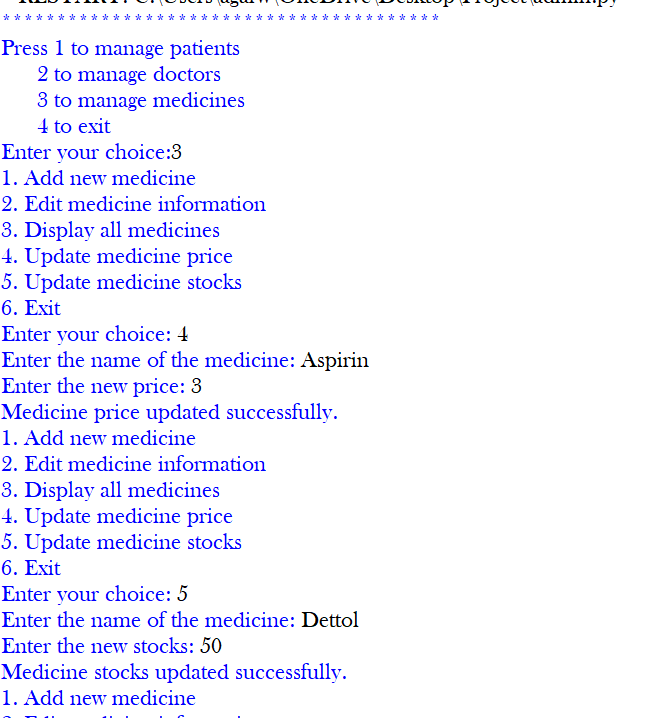


**Add new medicine**

**Update medicine price**

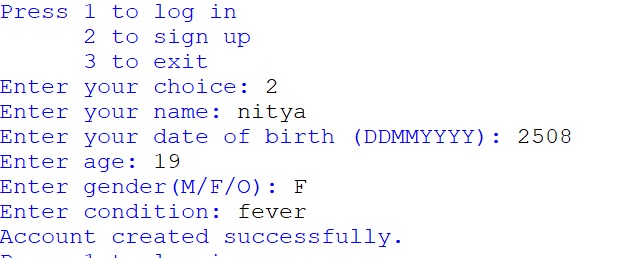
****

**Update medicine stocks**

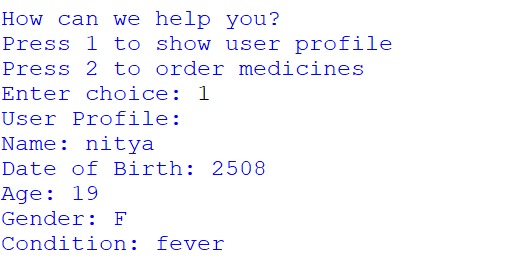
****

**2.User mode**

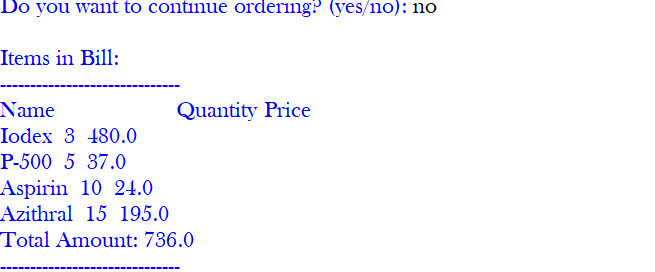
**Sign up**



**To show user profile**

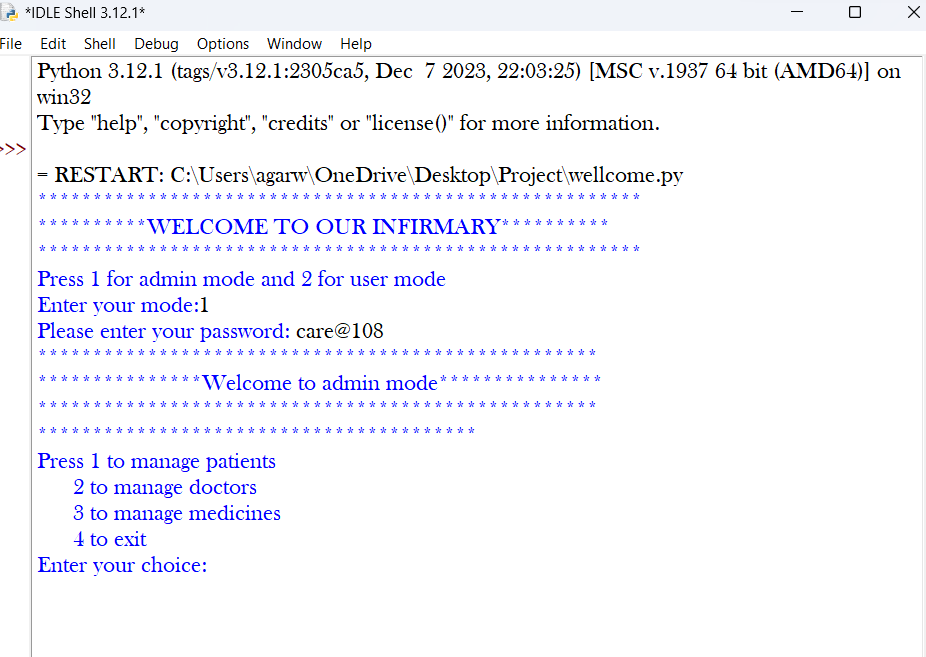


**Generate bill**



Key Features :-

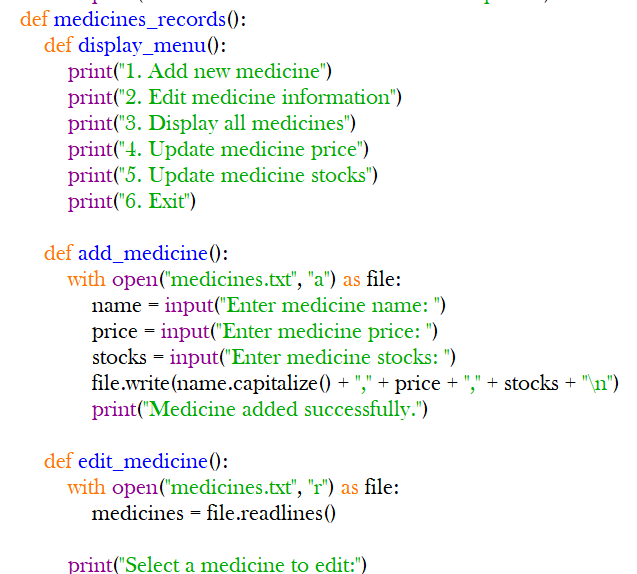
* **Authentication:**

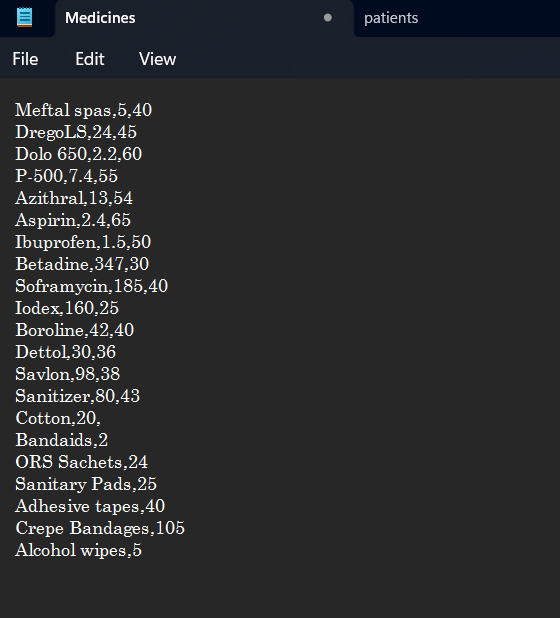




The program ensures secure access by requiring administrators to enter a password in admin mode as well as user mode.

* Data Management:





Patient records, doctor details, and medicine inventory are stored in separate text files and managed through file I/O operations.

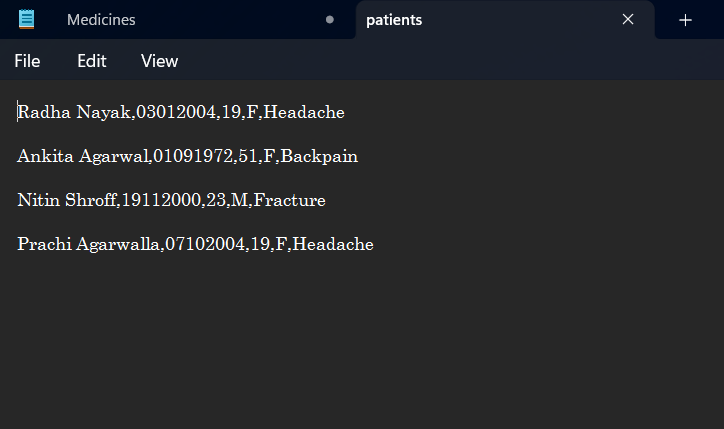
* Modular Design:



The program is modularized, with separate functions and modules for different functionalities, enhancing code organization and readability.

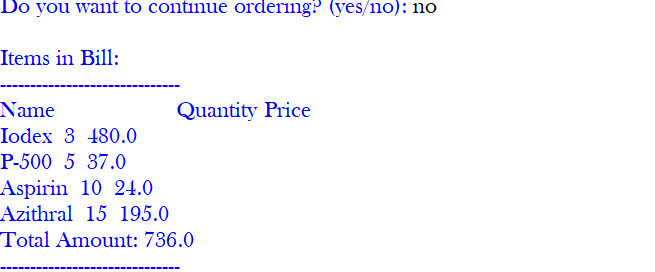
* Input Validation:





Various input validation checks are implemented to handle invalid inputs gracefully and provide a smooth user experience.

* Billing System:



Users can order medicines and generate a bill for their purchases, facilitating seamless transactions.

**CONCEPTS USED :-**

1. File Handling:

The program reads from and writes to different text files to store patient information, doctor information and medicine details, respectively. The open() function is used to open files, and the with statement ensures that the files are properly closed after use.

1. Functions and nested functions:

The **admin\_mode()** function contains nested functions (**patient\_management()**, **doctor\_management()**, **medicines\_records()**), which encapsulate different aspects of admin functionalities .The program defines several functions (**login()**, **signup()**, **show\_user\_profile()**, and **bill()**) to modularize the code and promote reusability. Functions help in organizing code and performing specific tasks.

1. Input/output:

The **input()** function is used to accept user input from the console. User input is validated and processed accordingly to perform actions like adding,editing,deleting details in the text files, logging in, signing up, and ordering medicines.

1. Conditional statements:

Conditional statements (**if**, **elif**, **else**) are used to execute different blocks of code based on certain conditions. For example, the script checks user input to determine whether to log in, sign up, or exit the program.

1. Loops:

The script uses **while** loops to repeatedly prompt the user for input until certain conditions are met. Nested loops are also used within the **bill()** function to handle user interactions related to ordering medicines. For example, the outer loop in **admin\_mode()** continues until the user chooses to exit, and inner loops within **patient\_management()**, **doctor\_management()**, and **medicines\_records()** handle the menu options for each functionality.

1. String Manipulation:

String manipulation techniques are used to process and format data. Functions like **split()** and string concatenation (**+** operator) are used to manipulate strings and extract relevant information from them. Other string manipulation techniques such as splitting, joining, and formatting are used to process and display data read from files, as well as to construct data to be written to files.

1. Variable scope:

The scope of variables is managed appropriately within functions and the main program.

1. Menu driven programs:

The menu-driven approach employed in the code allows users to interact with the program by presenting them with a series of options displayed as a menu. Users can navigate through different functionalities by selecting the corresponding menu options..

1. Data Structures:

Depending on the selected mode (admin or user), the script imports functionality from different modules (**admin** or **user**). This allows for modular code organization and separation of concerns, making the codebase more maintainable and scalable.

1. OOPS Concept:

* **Class:**

Classes are blueprints for creating objects. In this code, classes like `User`, `Patient`, `Medicine`, `Bill`, `Pharmacy`, `ManagementMode`, `Infirmary`, `PatientManagement`, `DoctorManagement`, and `MedicinesRecords` are defined. Each class encapsulates data (attributes) and methods (functions) to operate on that data. For example, the `User` class holds attributes like `name` and `dob`, along with methods such as `login` and `signup`.

* **Inheritance:**

Inheritance is a way to form new classes using classes that have already been defined. In this code, the `Patient` class inherits from the `User` class, allowing it to use the attributes and methods of the `User` class. Similarly, the `PatientManagement`, `DoctorManagement`, and `MedicinesRecords` classes inherit from the `ManagementMode` class, which allows them to utilize its methods.

* **Encapsulation:**

Encapsulation is the bundling of data (attributes) and the methods (functions) that operate on that data into a single unit, i.e., a class. Each class in this code encapsulates data and methods within itself. For instance, the `Bill` class encapsulates the `items` list and methods like `generate\_bill` and `add\_item`.

* **Polymorphism:**

Polymorphism allows objects to be treated as instances of their parent class. In this code, the methods `login` and `signup` are examples of polymorphism. Both methods exist in both the `User` and `Patient` classes, but their implementations are different.

* **Abstraction:**

Abstraction refers to hiding the complex details and showing only the necessary features of an object. In this code, the `ManagementMode` class is an abstract class providing a template for its subclasses, such as `PatientManagement`, `DoctorManagement`, and `MedicinesRecords`. It encapsulates the common functionality and structure, which is then inherited by its subclasses.