HEALTHCARE

HOSPITAL MANAGEMENT

Registration No	: 25BAS10098
Name of Student	:Bhavadharani.S
Course Name	: Introduction to Problem Solving and Programming
Course Code	: CSE1021
School Name	: SMEC
Slot	: B11+B12+B13
Class ID	: BL2025260100796
Semester	: FALL 2025/26

HOSPITAL MANAGEMENT

WHY HOSPITAL MANAGEMENT IS NEEDED

Hospital management is **extremely important** because it is the backbone that ensures the entire healthcare facility operates smoothly, efficiently, and, most importantly, provides **high-quality patient care**.

Without effective management, even the most skilled doctors and state-of-the-art equipment can struggle to deliver consistent care.

Here are the key reasons why hospital management is vital:

1. Enhanced Patient Care and Outcomes

- **Quality and Safety:** Management sets and enforces protocols for patient safety, infection control, and quality improvement programs, which directly reduce medical errors and complications.
- **Streamlined Processes:** They manage everything from appointment scheduling, patient registration, and lab services to discharge, reducing wait times and creating a smoother, less stressful patient experience.
- **Resource Allocation:** Management ensures that essential medical supplies, equipment, and medications are always available when needed for treatment.

2. Operational Efficiency and Sustainability

- **Financial Health:** Managers oversee budgeting, financial planning, billing, and cost control. This financial stability is crucial for the hospital's long-term survival and ability to invest in new technology or infrastructure.
- **Workflow Optimization:** They design efficient workflows for all departments, from clinical to administrative, which cuts down on wasted time and resources.
- **Technology Integration:** Management is responsible for adopting and implementing technology like Electronic Health Records (EHRs), which improves data accuracy, information access, and communication across the care team.

3. Staff Management and Support

- **Human Resources:** They handle recruitment, training, and retention of qualified doctors, nurses, and administrative staff, which is critical in maintaining a high level of expertise.
- **Morale and Productivity:** Good management fosters a positive work environment, addresses staff burnout, and ensures adequate staffing levels, allowing clinical professionals to focus on patient care.
- **Communication:** They facilitate seamless communication and collaboration between different departments and specialties.

4. Legal and Regulatory Compliance

- **Adherence to Law:** Managers ensure the hospital complies with complex healthcare laws, ethical standards, and government regulations.
- **Risk Management:** They develop policies to mitigate risks, protect patient data privacy (e.g., HIPAA), and maintain necessary accreditation and certifications.

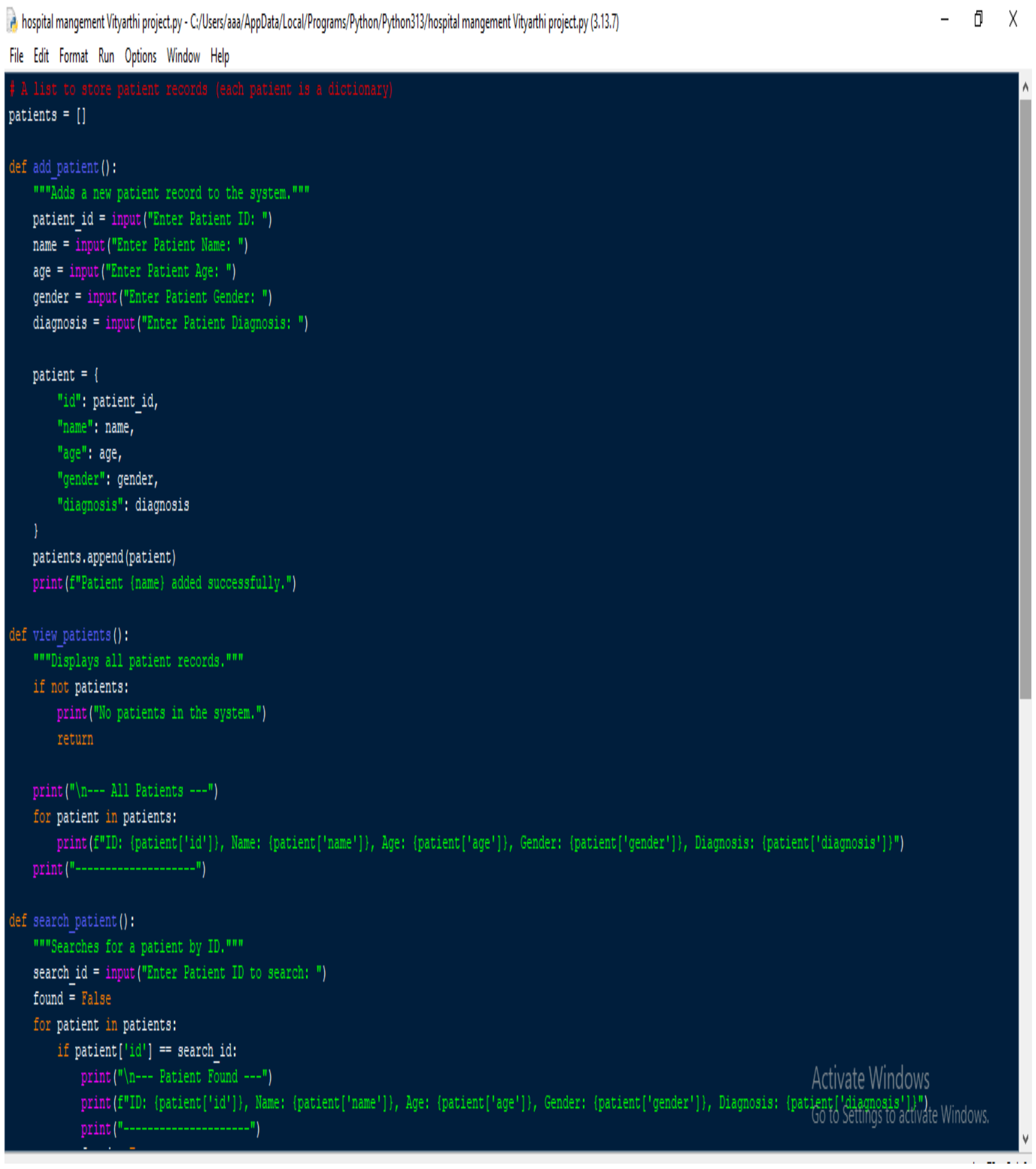
In short, **hospital management is the engine** that keeps the entire system running, ensuring that the hospital is not just a place for medical services, but a safe, efficient, and financially sustainable institution that can consistently deliver the best possible care to the community.

SOLUTION:

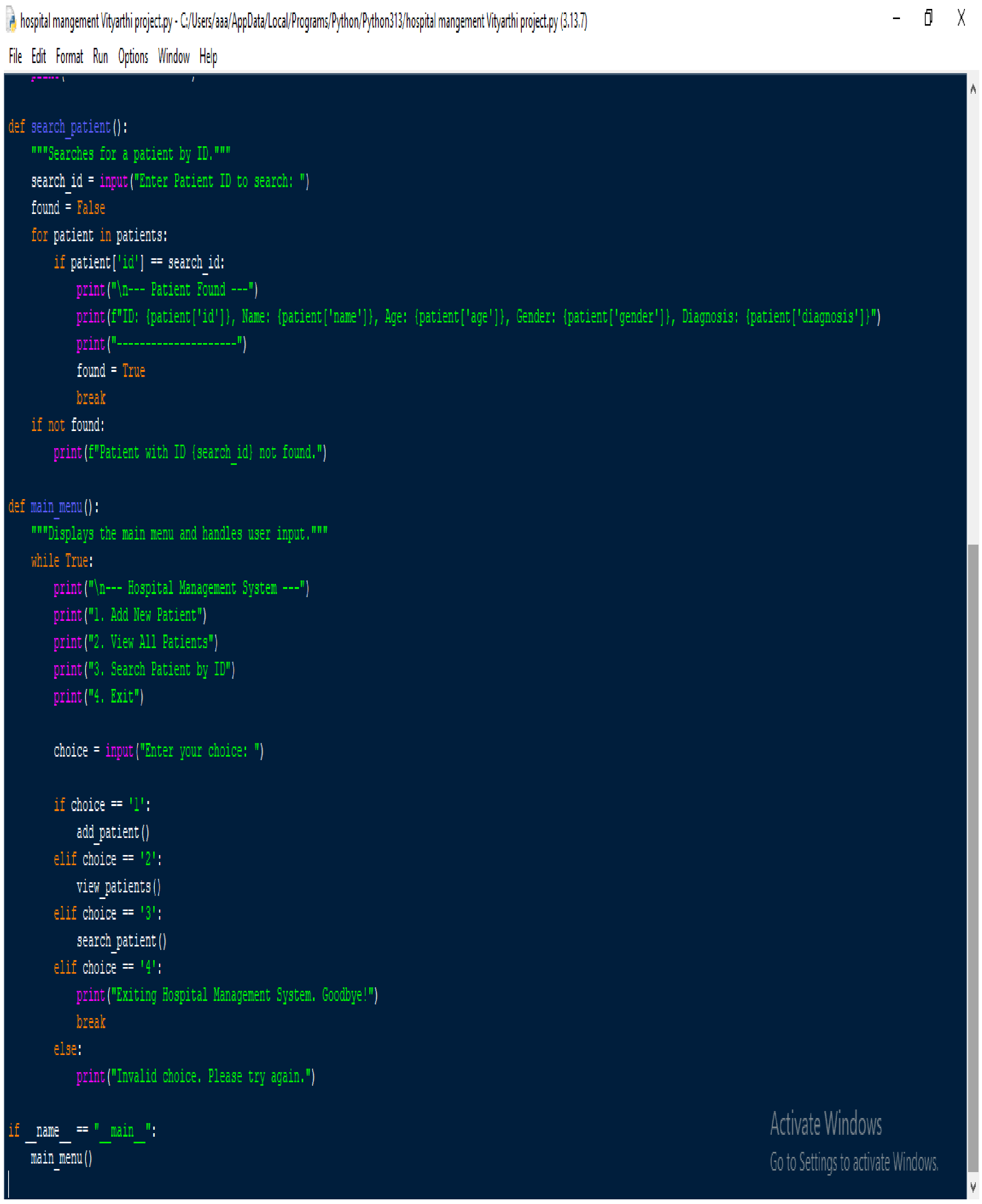
Making a simple python project to store all the patients details with ease

SOURCE CODE:

SCREEN SHOTS


hospital mangement Vityarthi project.py - C:/Users/aaa/AppData/Local/Programs/Python/Python313/hospital mangement Vityarthi project.py (3.13.7)
File Edit Format Run Options Window Help
Activate Windows
Go to Settings to activate Windows.

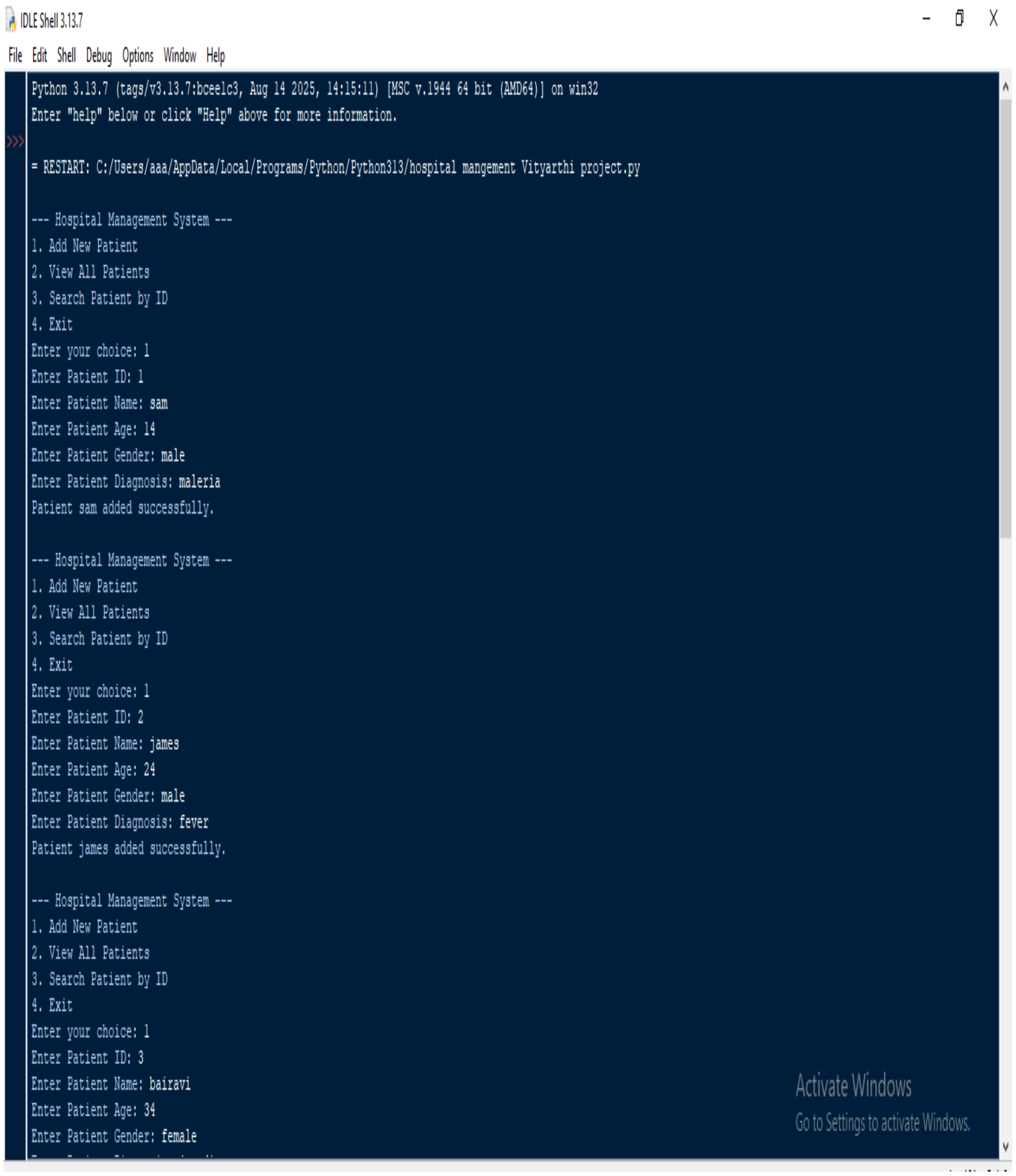
```python
# A list to store patient records (each patient is a dictionary)
patients = []

def add_patient():
    """Adds a new patient record to the system."""
    patient_id = input("Enter Patient ID: ")
    name = input("Enter Patient Name: ")
    age = input("Enter Patient Age: ")
    gender = input("Enter Patient Gender: ")
    diagnosis = input("Enter Patient Diagnosis: ")

    patient = {
        "id": patient_id,
        "name": name,
        "age": age,
        "gender": gender,
        "diagnosis": diagnosis
    }
    patients.append(patient)
    print(f"Patient {name} added successfully.")

def view_patients():
    """Displays all patient records."""
    if not patients:
        print("No patients in the system.")
        return

    print("\n--- All Patients ---")
    for patient in patients:
        print(f"ID: {patient['id']}, Name: {patient['name']}, Age: {patient['age']}, Gender: {patient['gender']}, Diagnosis: {patient['diagnosis']}")
    print("--------------------")

def search_patient():
    """Searches for a patient by ID."""
    search_id = input("Enter Patient ID to search: ")
    found = False
    for patient in patients:
        if patient['id'] == search_id:
            print("\n--- Patient Found ---")
            print(f"ID: {patient['id']}, Name: {patient['name']}, Age: {patient['age']}, Gender: {patient['gender']}, Diagnosis: {patient['diagnosis']}")
            print("--------------------")
```


hospital mangement Vityarthi project.py - C:/Users/aaa/AppData/Local/Programs/Python/Python313/hospital mangement Vityarthi project.py (3.13.7)
File Edit Format Run Options Window Help
Activate Windows
Go to Settings to activate Windows.

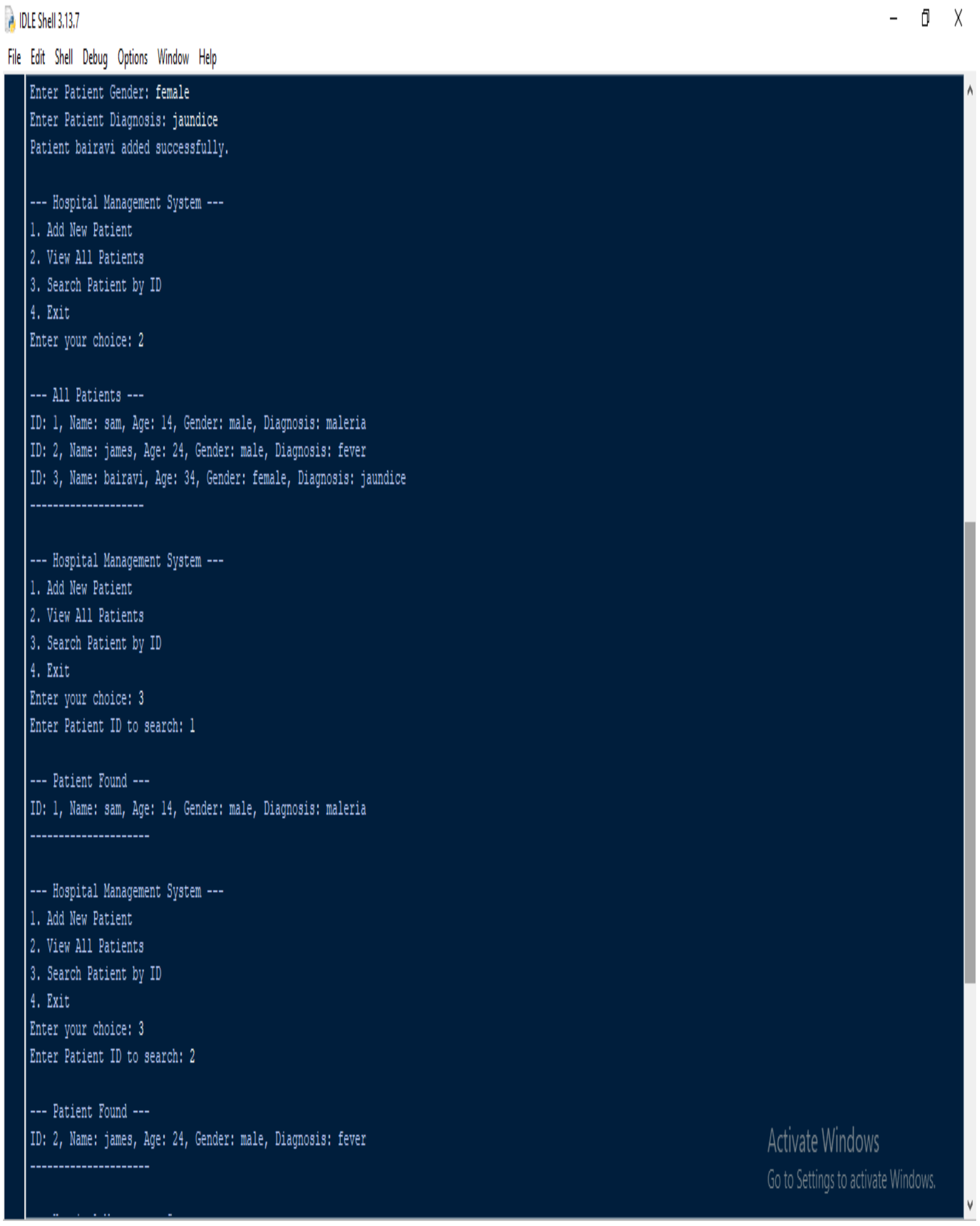
```python
def search_patient():
    """Searches for a patient by ID."""
    search_id = input("Enter Patient ID to search: ")
    found = False
    for patient in patients:
        if patient['id'] == search_id:
            print("\n--- Patient Found ---")
            print(f"ID: {patient['id']}, Name: {patient['name']}, Age: {patient['age']}, Gender: {patient['gender']}, Diagnosis: {patient['diagnosis']}")
            print("--------------------")
            found = True
            break
    if not found:
        print(f"Patient with ID {search_id} not found.")

def main_menu():
    """Displays the main menu and handles user input."""
    while True:
        print("\n--- Hospital Management System ---")
        print("1. Add New Patient")
        print("2. View All Patients")
        print("3. Search Patient by ID")
        print("4. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            add_patient()
        elif choice == '2':
            view_patients()
        elif choice == '3':
            search_patient()
        elif choice == '4':
            print("Exiting Hospital Management System. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main_menu()
```

RESULT:

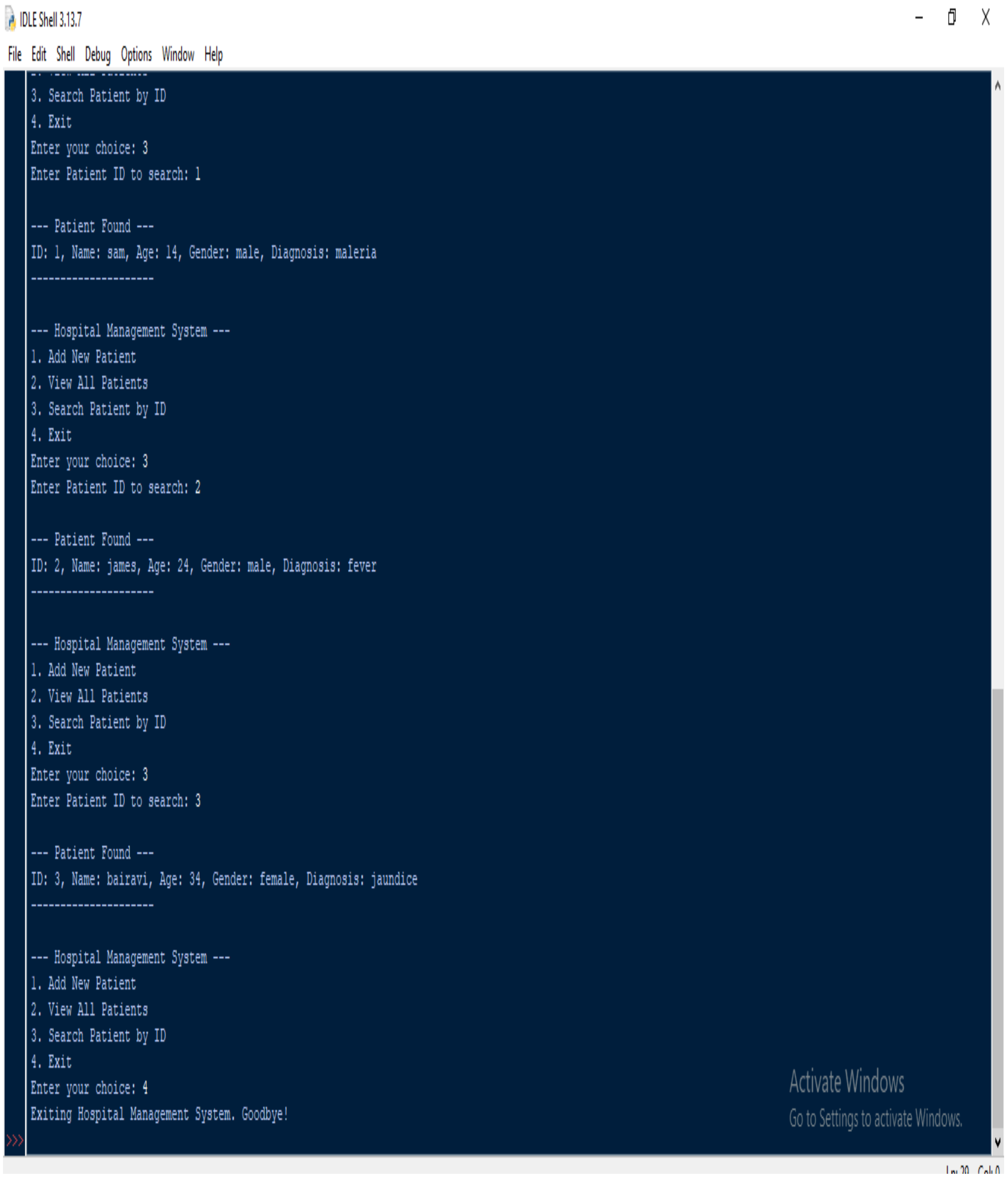
SCREEN SHOTS


IDLE Shell 3.13.7
File Edit Shell Debug Options Window Help
Activate Windows
Go to Settings to activate Windows.

```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
= RESTART: C:/Users/aaa/AppData/Local/Programs/Python/Python313/hospital mangement Vityarthi project.py

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 1
Enter Patient ID: 1
Enter Patient Name: sam
Enter Patient Age: 14
Enter Patient Gender: male
Enter Patient Diagnosis: maleria
Patient sam added successfully.

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 1
Enter Patient ID: 2
Enter Patient Name: james
Enter Patient Age: 24
Enter Patient Gender: male
Enter Patient Diagnosis: fever
Patient james added successfully.

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 1
Enter Patient ID: 3
Enter Patient Name: bairavi
Enter Patient Age: 34
Enter Patient Gender: female
```



```
Enter Patient Gender: female
Enter Patient Diagnosis: jaundice
Patient bairavi added successfully.

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 2

--- All Patients ---
ID: 1, Name: sam, Age: 14, Gender: male, Diagnosis: maleria
ID: 2, Name: james, Age: 24, Gender: male, Diagnosis: fever
ID: 3, Name: bairavi, Age: 34, Gender: female, Diagnosis: jaundice
--------------------

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 3
Enter Patient ID to search: 1

--- Patient Found ---
ID: 1, Name: sam, Age: 14, Gender: male, Diagnosis: maleria
--------------------

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 3
Enter Patient ID to search: 2

--- Patient Found ---
ID: 2, Name: james, Age: 24, Gender: male, Diagnosis: fever
--------------------
```


IDLE Shell 3.13.7

File Edit Shell Debug Options Window Help

```
3. Search Patient by ID
4. Exit
Enter your choice: 3
Enter Patient ID to search: 1

--- Patient Found ---
ID: 1, Name: sam, Age: 14, Gender: male, Diagnosis: maleria
--------------------

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 3
Enter Patient ID to search: 2

--- Patient Found ---
ID: 2, Name: james, Age: 24, Gender: male, Diagnosis: fever
--------------------

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 3
Enter Patient ID to search: 3

--- Patient Found ---
ID: 3, Name: bairavi, Age: 34, Gender: female, Diagnosis: jaundice
--------------------

--- Hospital Management System ---
1. Add New Patient
2. View All Patients
3. Search Patient by ID
4. Exit
Enter your choice: 4
Exiting Hospital Management System. Goodbye!
>>>
```

Functional Requirements

---

1. Patient Data Management

- **FR1.1: Add Patient Record**
  - The system must allow a user to **input and store** a new patient record, including their ID, Name, Age, Gender, and Diagnosis. (Implemented in add_patient())
- **FR1.2: Store Patient Records**
  - The system must be able to **store multiple patient records** in a list structure. (Using the global patients list and dictionaries)

---

2. Information Retrieval

- **FR2.1: View All Patient Records**
  - The system must be able to **display a complete list** of all stored patient records with their ID, Name, Age, Gender, and Diagnosis. (Implemented in view_patients())
  - The system must **notify the user** if there are no patients in the system.
- **FR2.2: Search Patient by ID**
  - The system must allow a user to **search** for a specific patient record using their unique Patient ID. (Implemented in search_patient())
  - The system must **display the details** of the found patient.
  - The system must **notify the user** if the patient ID is not found.

---

3. User Interface and Control

- **FR3.1: Display Main Menu**
  - The system must **present a menu** with the following options: Add New Patient, View All Patients, Search Patient by ID, and Exit. (Implemented in main_menu())
- **FR3.2: Process User Choice**

    - The system must **accept and execute** the corresponding function based on the user's menu choice (1, 2, or 3).
- **FR3.3: Exit System**
    - The system must allow the user to **terminate the program** (Option 4).
- **FR3.4: Handle Invalid Input**
    - The system must **handle invalid menu choices** by notifying the user and redisplaying the menu.

Non-Functional Requirements

1. Usability

- **Clarity:** The system's menu and input prompts are clear, guiding the user on the next action.
- **Intuitiveness:** The system is designed with a simple command-line menu (1, 2, 3, 4), making it relatively easy for a user to operate without extensive training.

2. Performance & Efficiency

- **Responsiveness:** For the current small scale, the system is expected to perform the operations (add, view, search) **instantaneously** with minimal delay.
- **Execution Time:** The time taken to execute core functions (like searching the patients list) must be negligible, given the small data size.

3. Reliability & Maintainability

- **Availability (Basic):** The core functions (add_patient, view_patients, etc.) are readily available when the application is running.
- **Portability:** The code, being standard Python, is **highly portable** and can run on any system with a Python interpreter installed.
- **Maintainability:** The code is structured with clear functions (def add_patient():, etc.), making it **easy to read and modify** for future enhancements.

4. Security (Minimal)

- **No Data Persistence:** A key non-functional **limitation** is that the data is **volatile**; it is only stored in memory (patients = []). There is no requirement for saving data to a file or database, meaning all data is lost when the program exits. This is a lack of the NFR **Data Persistence**.
- **No Access Control:** There is **no authentication or authorization** mechanism implemented. Any user can access and modify all patient records.

The most important non-functional requirements that would need to be *added* if this system were to be used in a real hospital would be **Security (Access Control)**, **Data Persistence**, and robust **Error Handling/Data Validation**.

The term "System Architecture" usually refers to the overall structure of software, including its components, their interrelationships, and the principles guiding its design.

For this small Python program, we don't have a complex, multi-layered, or distributed architecture. Instead, the architecture is best described using the **Monolithic** and **Procedural** styles.

system architecture :

1. Monolithic Structure

- The entire application—data storage, business logic, and user interface—is contained within a **single executable file/module** (.py file).
- All components (functions) run within the same process space and share the same global state (patients list).

2. Components and Tiers (Three-Tier Abstraction)

Although the code is simple, we can mentally separate its responsibilities into the three standard logical tiers of an application:

Architectural Tier	Description	Code Component

Presentation Layer (User Interface)	Handles user input and displays results via the console.	main_menu(), input() and print() statements in all functions.
Business Logic Layer (Application Logic)	Contains the rules for managing the patient data (CRUD operations).	add_patient(), view_patients(), search_patient().
Data Layer (Data Access/Storage)	Responsible for storing and retrieving the system's core data.	The global list **patients** (In-Memory Storage).

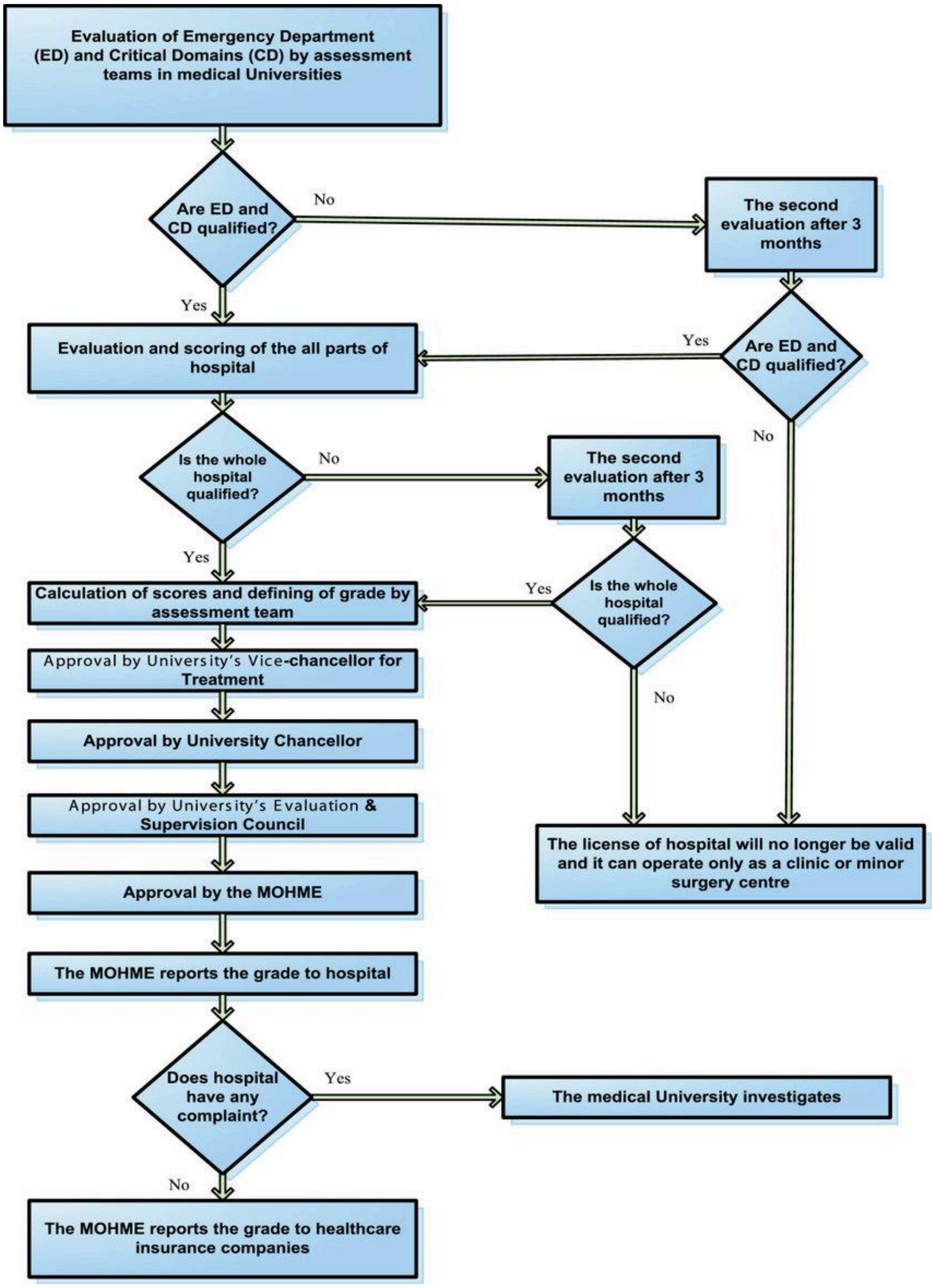
3. Procedural Flow and Relationships

The program follows a clear procedural flow:

1. **Start:** Execution begins with if __name__ == "__main__": which calls main_menu().
2. **Control Loop:** main_menu() provides a continuous loop (while True) that manages the application's state.
3. **Function Calls:** Based on the user's choice, main_menu() directly calls the appropriate function (add_patient, view_patients, or search_patient).
4. **Global Data Access:** All logic functions (add_patient, etc.) directly access and modify the shared, global **patients** list.

HOW HOSPITALS ARE MANAGED

FLOW CHART



Evaluation of Emergency Department (ED) and Critical Domains (CD) by assessment teams in medical Universities
Are ED and CD qualified?
No
The second evaluation after 3 months
Yes
Evaluation and scoring of the all parts of hospital
Yes
Are ED and CD qualified?
No
Is the whole hospital qualified?
No
The second evaluation after 3 months
Yes
Calculation of scores and defining of grade by assessment team
Yes
Is the whole hospital qualified?
No
Approval by University's Vice-chancellor for Treatment
Approval by University Chancellor
Approval by University's Evaluation & Supervision Council
The license of hospital will no longer be valid and it can operate only as a clinic or minor surgery centre
Approval by the MOHME
The MOHME reports the grade to hospital
Does hospital have any complaint?
Yes
The medical University investigates
No
The MOHME reports the grade to healthcare insurance companies

1. Data Structure Implementation

The system uses the simplest possible approach for temporary data storage:

- **Global List:** A list named patients is initialized outside all functions (patients = []). This list serves as the **in-memory database** for the entire application.
- **Dictionary Records:** Each patient record is stored as a Python dictionary within the patients list.
- **Schema (Fields):** Each dictionary adheres to a fixed schema with five string fields: "id", "name", "age", "gender", and "diagnosis".

Limitation: Since this data is stored only in RAM (in-memory), all records are lost the moment the program terminates.

2. Core Functional Implementations (Business Logic)

The program implements three primary operations based on the Create, Read, Update, Delete (CRUD) paradigm, though it lacks the "Update" and "Delete" capabilities.

A. add_patient() (Create/Input)

- **Input Gathering:** Uses the built-in input() function sequentially to prompt the user for the five required patient details.
- **Data Type: Crucially, all data is captured as strings.** No attempt is made to cast age to an integer or validate the type of the patient_id.
- **Record Creation:** A new dictionary (patient) is constructed using the gathered string inputs.
- **Persistence (In-Memory):** The new dictionary is appended directly to the global patients list using patients.append(patient).

B. view_patients() (Read/Display All)

- **Empty Check:** A primary check (if not patients:) prevents unnecessary execution and informs the user if the list is empty.
- **Iteration:** It uses a simple for loop to iterate through every dictionary stored in the global patients list.
- **Output Formatting:** Inside the loop, it uses an f-string to format and print the details of each patient record to the console.

C. search_patient() (Read/Display Specific)

- **Search Input:** Prompts the user for the search_id string.
- **Linear Search:** It performs a simple **linear search** by iterating through the entire patients list.
- **Found Flag:** A boolean flag (found = False) is used to track whether a match was located.
- **Break on Match:** If patient['id'] == search_id is true, the record is printed, the found flag is set to True, and the break statement immediately exits the loop, optimizing performance once the single desired record is found.
- **Not Found Logic:** After the loop completes, the state of the found flag determines if the "not found" message is displayed.

3. Control Flow Implementation

The entire user experience and application control are managed by the main_menu() function.

- **Execution Entry Point:** The standard Python conditional if __name__ == "__main__": ensures that main_menu() is the first function called when the script is run directly.
- **Main Control Loop:** The core is an infinite while True loop, ensuring the menu is repeatedly displayed after every action until the user explicitly chooses to exit.
- **User Interface:** The menu options are presented using print(), and the user's input is captured via input().
- **Command Dispatch:** An if/elif/else chain handles the command dispatch:
    - Choices '1', '2', and '3' call their respective functional routines.
    - Choice '4' uses the break statement to exit the while True loop, which ends the program.
    - The else block handles any invalid input, printing an error message and continuing the loop.

Testing Approach

The primary strategy is **Manual Black-Box Testing**. This means the tester interacts with the system solely through the command line (the input/output interface) without looking directly at the code implementation details, aiming to ensure all features meet the implied functional requirements

Challenges Faced

- Inefficiency in loops and not optimizing for performance, which can cause slow or time-consuming execution for high input values
- building functions with correct logic
- testing
- understanding the problem and requirements
- debugging

Learnings & Key Takeaways

- Modularity (via Funct
- Control (via Main Menu Loop)
- Modeling (Data Structure: List of Dictionaries)
- Global (State Management Challenge)
- Persistence (Data Loss, Lack of File/Database)
- Validation (Missing Input Checks)
- Integrity (Missing Unique ID Checks)
- Performance (Linear Search Inefficiency)
- Optimization (break statement use)

References:

WHO (WORLD HEALTH ORGANISATION)

RESEARCH PAPERS