LAB EXERCISE: Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.

ANS :-

Java :-

```
Public class Demo

{

        Public Static void main(String [] args)

        {

                System.out.println("Hello World");

        }

}
```

Note:

- Java code must be inside a class.
- The main method is the program's entry point.
- Semicolons (;) are required at the end of each statement.

Structure :-

1.Declares a public class named HelloWorld. In Java, every program must be inside a class.

2. This is the **main method**, where the program starts executing.

3. Prints the message "Hello, World!" to the console.

Python :-

```
Print("Hello world ")
```

Note:

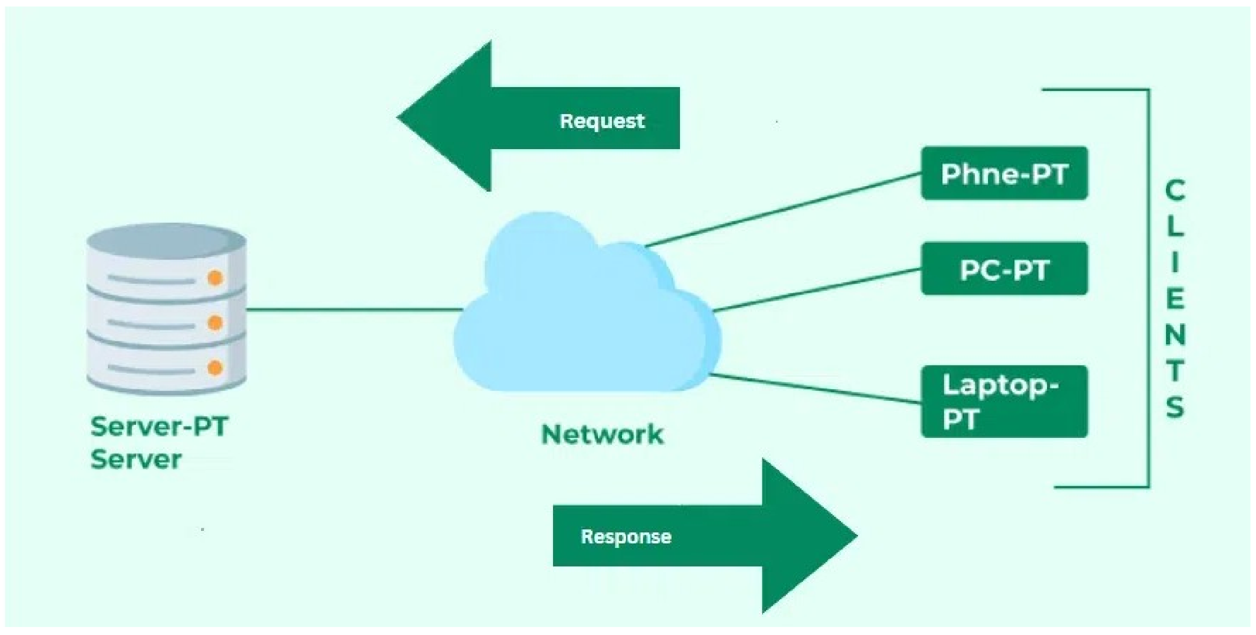
- Python is very simple.
- No need for a main method or class.
- No semicolons (;) are required.

Structure :-

1. This line uses the built-in print() function to display the message "Hello, World!" on the screen.

LAB EXERCISE: Research and create a diagram of how data is transmitted from a client to a server over the internet.

Diagram of Client-server Architecture :-



Client and Server Explained

Client

- A client is any device or software application that initiates communication by sending requests to a server for data or services. Clients depend on servers to provide the content or functionality they need.
- Clients are usually operated by end users and are designed to be user-friendly and lightweight compared to servers.

Common Client Applications:

- **Web Browsers** – e.g., Google Chrome, Mozilla Firefox (used to access websites)
- **Email Applications** – e.g., Gmail, Microsoft Outlook (used to send/receive emails)
- **Mobile Apps** – e.g., Instagram, WhatsApp (used to interact with backend services)

Server

- A server is a powerful computer system or software program that waits for requests from clients and responds with the required data or service. Servers are optimized for performance, reliability, and the ability to handle multiple client connections at the same time.
- Servers run continuously and are often hosted in data centers or cloud platforms.

Common Server Types:

- **Web Servers** – e.g., Apache, Nginx (serve web pages to browsers)
- **Email Servers** – handle sending, receiving, and storing emails
- **Database Servers** – e.g., MySQL, PostgreSQL (manage and provide access to databases)
- **File Servers** – allow users to store and access files over a network

LAB EXERCISE: Research different types of internet connections (e.g., broadband, fiber, satellite)and list their pros and cons.

1. Dial-Up Connection

- Uses a modem and telephone line to connect to the internet.
- Very slow and outdated.
- Cannot use phone and internet at the same time.
- Must dial a number to connect.

2. Broadband Connection

- High-speed internet access.
- Can work over cable or telephone lines.
- Supports multiple users at once.
- No need to dial, always connected.
- Can use internet and phone together.

3. DSL (Digital Subscriber Line)

Provides internet through telephone lines.

Always-on connection (no dialing needed).

Speed: 128 Kbps to 8 Mbps.

Can use internet and landline simultaneously.

4. Cable Connection

- Uses cable TV lines for internet access.
- Requires a cable modem.
- Speed: 512 Kbps to 20 Mbps.
- Fast for both downloads and uploads.

5. Satellite Connection

- Used in rural or remote areas.
- Connects via satellites in Earth's orbit.
- Speed: 512 Kbps to 2 Mbps.
- Higher delay/latency due to long-distance signals.

6. Wireless Connection Uses radio frequency to connect.

- No cables or telephone lines needed.
- Speed: 5 Mbps to 20 Mbps.
- Can be accessed from anywhere within range.

7. Cellular Connection (3G/4G)

- Internet via mobile networks (3G, 4G).
- 3G speed: ~2 Mbps, 4G: ~21 Mbps.
- No wires required, portable.
- Depends on mobile service provider.

8. ISDN (Integrated Services Digital Network)

- Uses digital telephone lines.
- Transmits voice and data.
- Speed: Up to 128 Kbps.
- Better quality than older dial-up systems.

LAB EXERCISE: Identify and explain three common application security vulnerabilities. Suggestpossible solutions.

1. SQL Injection (SQLi)

- **What it is:**
  Attackers insert malicious SQL code into input fields to manipulate databases, potentially exposing or destroying data.
- **Example:**
  Entering ' OR '1'='1 in a login form to bypass authentication.
- **Possible Solutions:**
  - Use **parameterized queries** or **prepared statements**.
  - Validate and sanitize all user inputs.
  - Implement least privilege access on the database.

---

2. Cross-Site Scripting (XSS)

- **What it is:**
  Attackers inject malicious scripts (usually JavaScript) into web pages viewed by other users, potentially stealing cookies or session tokens.
- **Example:**
  Injecting <script>alert('Hacked!')</script> in a comment box.
- **Possible Solutions:**
  - Escape or encode output data before rendering in the browser.
  - Use Content Security Policy (CSP) headers.
  - Sanitize user inputs with libraries or frameworks.

3. Broken Authentication

- **What it is:**
  Weak authentication mechanisms allow attackers to compromise user accounts through brute force, credential stuffing, or session hijacking.
- **Example:**
  Using weak passwords or exposing session IDs in URLs.
- **Possible Solutions:**
  - Enforce strong password policies and multi-factor authentication (MFA).
  - Use secure, HTTP-only cookies for session management.
  - Implement account lockouts or rate limiting on login attempts.

LAB EXERCISE: Identify and classify 5 applications you use daily as either system software or application software.

Windows OS / macOS / Linux

- **Type:** System Software
- **Explanation:** Operating system that manages hardware and basic functions of the computer.

Google Chrome (Web Browser)

- **Type:** Application Software
- **Explanation:** Software used to browse the internet, runs on top of the operating system.

Microsoft Word

- **Type:** Application Software
- **Explanation:** Program used for creating and editing documents.

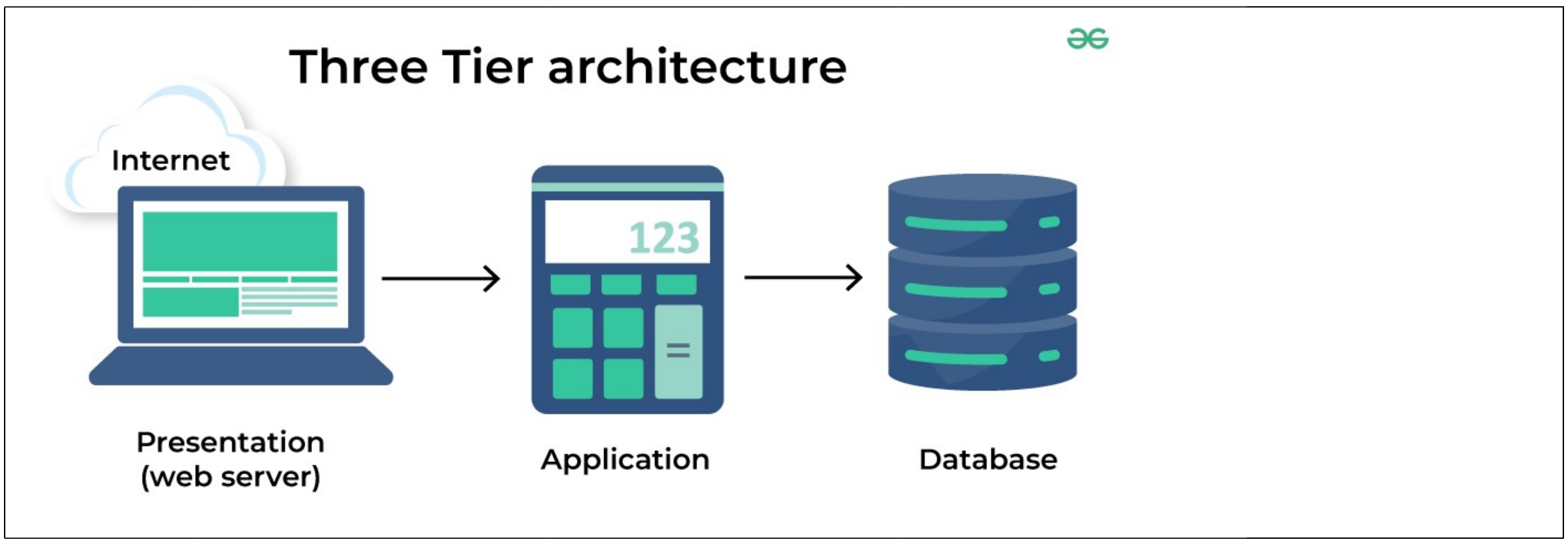
Antivirus Software (e.g., Windows Defender)

- **Type:** System Software
- **Explanation:** Software that protects the system by detecting and removing malware.

File Explorer (Windows) / Finder (Mac)

- **Type:** System Software
- **Explanation:** Tool integrated into the operating system to manage and access files and folders.

LAB EXERCISE: Design a basic three-tier software architecture diagram for a web application.



A **three-tier architecture** separates the application into three layers:

1. **Presentation Tier (Client Layer):**
   - The user interface where users interact with the app.
   - Usually a web browser or mobile app.
2. **Logic Tier (Application Server):**
   - Contains the business logic and processes user requests.
   - Handles communication between the presentation and data tiers.
3. **Data Tier (Database Server):**
   - Responsible for storing, retrieving, and managing data.
   - Usually a database like MySQL, PostgreSQL, or MongoDB.

LAB EXERCISE: Create a case study on the functionality of the presentation, business logic, and dataaccess layers of a given software system.

1. Presentation Layer

- This is what the user sees and uses.
- Example: The website pages where you search for books, add them to the cart, and place orders.

2. Business Logic Layer

- This layer makes decisions and applies rules.
- Example: It checks if the book is in stock, calculates discounts, and verifies your payment.

3. Data Access Layer

- This layer talks to the database.
- Example: It saves your order, gets book details, and updates the stock after a purchase.

LAB EXERCISE: Explore different types of software environments (development, testing, production).Set up a basic environment in a virtual machine

Development Environment:

- This is where application or system development tasks such as **designing**, **programming**, **debugging**, and initial testing take place. Developers write and test their code in this environment before moving it forward for more structured testing.

Test Environment:

- As the name suggests, this is where the application is tested to **identify and fix errors**. QA teams perform various tests like functional, integration, or regression testing to ensure the application works as expected.

Staging Environment:

- This is a pre-production environment where all the work done in the development environment is merged and built into a complete system. It is often used for final testing, including automated build processes, before moving the software to production. It closely mirrors the production environment.

Production Environment:

- This is the final environment in the software development process. New builds or updates that have passed testing and staging are deployed here for end users. It must be stable, secure, and fully functional, as it handles real user interactions and live data.

Set up a basic environment in a virtual machine

Step 1: Install VirtualBox

- Download and install **VirtualBox** from the official website:
  https://www.virtualbox.org

Step 2: Download Ubuntu ISO

- Download the latest version of **Ubuntu Desktop ISO** from:
  https://ubuntu.com/download/desktop

Step 3: Create a New Virtual Machine

1. Open VirtualBox → Click on **"New"**
2. Name: My_Dev_VM
3. Type: Linux, Version: Ubuntu (64-bit)
4. Assign RAM: At least **2048 MB (2 GB)**
5. Create a virtual hard disk: Select **20 GB** or more

Step 4: Install Ubuntu in the VM

1. Start the new VM
2. Select the downloaded Ubuntu ISO file
3. Follow on-screen instructions to install Ubuntu (choose language, keyboard layout, etc.)

Step 5: Set Up the Development Environment

1. Once Ubuntu is installed and running:
2. Open the terminal and install basic tools:

```
sudo apt update
sudo apt install -y build-essential python3 git curl
```

LAB EXERCISE: Write and upload your firstsource code file to Github

Step 1: Write Your First Code File

Create a simple "Hello World" program on your computer. You can use any language.
Example (in Python):

Create a file named: hello.py

```
print("Hello, GitHub!")
```

Save the file.

Step 2: Create a GitHub Account (if you don't have one)

- Go to https://github.com
- Sign up or log in

Step 3: Create a New Repository

1. Click the **+** icon (top right) → Select **"New repository"**
2. Name it something like first-code
3. Add a description (optional)
4. Keep it **Public**

5. **Do not check** "Initialize with a README" (for now)
6. Click **Create repository**

Step 4: Upload Your Code File

1. After the repo is created, click **"Uploading an existing file"**
2. Drag and drop your hello.py file or click **"choose your files"**
3. Scroll down and click **"Commit changes"**

Your file is now uploaded to GitHub!

LAB EXERCISE: Create a Github repository and document how to commit and push code changes

Step 1: Log in to GitHub

- Go to https://github.com
- Log in to your account

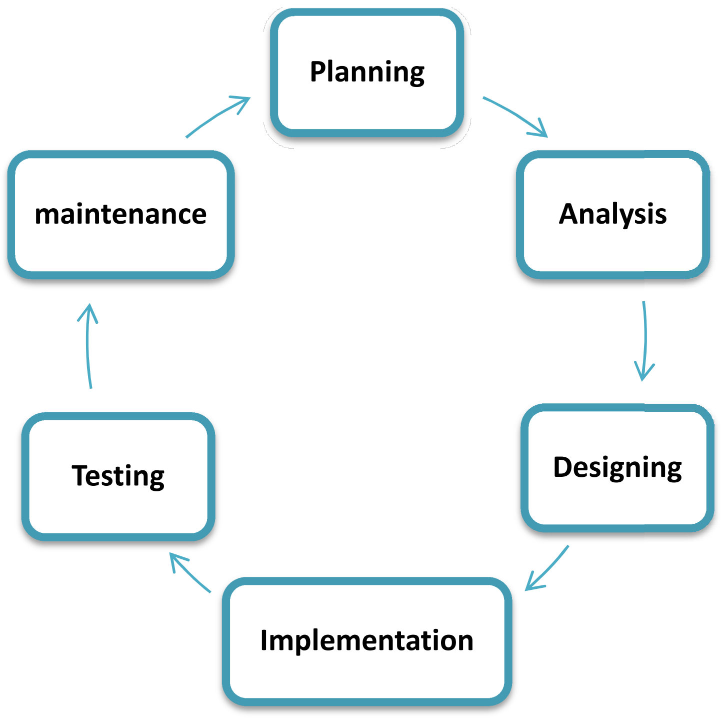
Step 2: Create a New Repository

1. Click the **"+" icon** on the top-right corner
2. Select **"New repository"**
3. Fill in the following:
    - **Repository name**: my-first-code
    - (Optional) Add a description
    - Choose **Public**
    - Check the box for **"Add a README file"**
4. Click **"Create repository"**

Step 3: Upload Your Code (Drag & Drop Method)

1. Inside your new repository, click the "Add file" button
2. Select "Upload files"
3. Drag and drop your code file(s) (e.g., hello.py) into the upload area
   OR click "choose your files" to browse manually
4. Scroll down to the "Commit changes" section
    - Add a commit message like: Added hello.py
    - Leave the other settings as they are
5. Click "Commit changes"

LAB EXERCISE: Create a flowchart representing the Software Development Life Cycle (SDLC).



LAB EXERCISE: Perform a functional analysis for an online shopping system.

1. User Registration and Authentication

- Users can create an account by providing personal details (name, email, password).
- Users can log in and log out securely.
- Password recovery and verification through email.

2. Product Catalog

- Display a list of products with details like name, price, description, and images.
- Users can search products by name, category, or filters (price range, brand).
- Sort products by popularity, price, or ratings.

3. Shopping Cart

- Users can add products to the shopping cart.
- Modify quantities or remove products from the cart.
- View the total price including taxes and discounts.

4. Order Placement

- Users can place an order by providing shipping address and payment details.
- Choose payment methods (credit card, PayPal, COD, etc.).
- Generate an order summary and confirmation.

5. Order Tracking

- Users can track the status of their orders (processing, shipped, delivered).
- Receive notifications/updates about the order status.

6. Payment Processing

- Secure processing of payments.
- Integration with payment gateways for various payment options.
- Confirmation of successful or failed transactions.

7. User Profile Management

- Users can view and update personal details.
- Manage saved addresses and payment methods.
- View past orders and reorder items.

8. Admin Functions

- Add, update, or remove products.
- Manage user accounts and orders.
- View sales reports and analytics.
- Handle customer support and complaints.

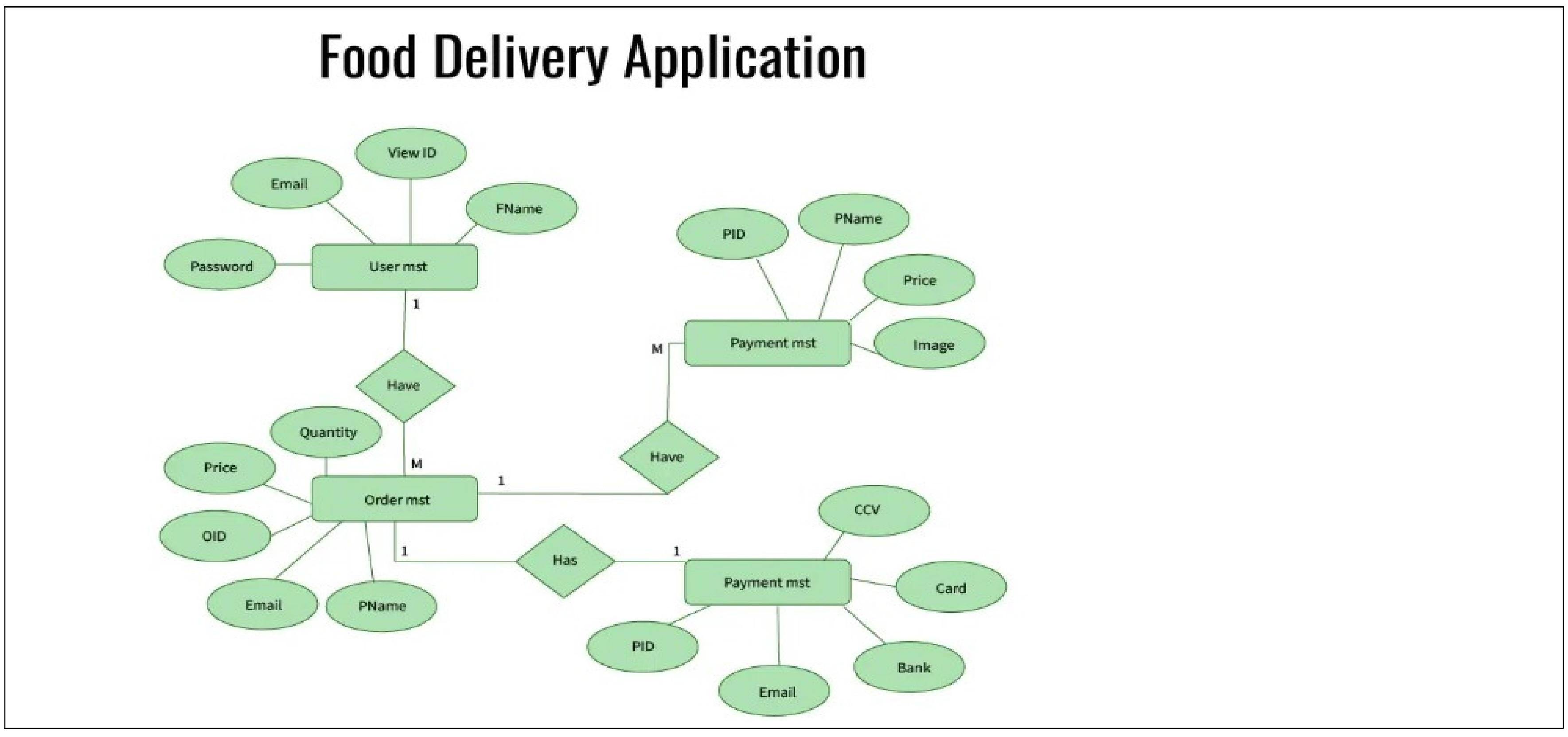
9. Reviews and Ratings

- Users can submit reviews and ratings for products.
- Display average ratings and customer feedback on product pages.

10. Security Features

- Data encryption for sensitive information.
- Protection against unauthorized access.
- Secure login and session management.

LAB EXERCISE: Design a basic system architecture for a food delivery app.



LAB EXERCISE: Develop test cases for a simple calculator program.

Test Case 1:
Operation: Addition
Input 1: 10
Input 2: 5
Expected Output: 15
Description: Adding two positive numbers.

Test Case 2:
Operation: Addition
Input 1: -3
Input 2: 7
Expected Output: 4
Description: Adding a negative and a positive number.

Test Case 3:
Operation: Subtraction
Input 1: 20
Input 2: 8
Expected Output: 12
Description: Subtracting smaller number from bigger number.

Test Case 4:
Operation: Subtraction
Input 1: 5
Input 2: 10
Expected Output: -5
Description: Subtraction resulting in a negative number.

Test Case 5:
Operation: Multiplication
Input 1: 4
Input 2: 6
Expected Output: 24
Description: Multiplying two positive numbers.

Test Case 6:
Operation: Multiplication
Input 1: -2
Input 2: 9
Expected Output: -18
Description: Multiplying a negative and a positive number.

Test Case 7:
Operation: Division
Input 1: 30
Input 2: 6
Expected Output: 5
Description: Dividing two positive numbers.

Test Case 8:
Operation: Division
Input 1: 7
Input 2: 2
Expected Output: 3.5
Description: Division resulting in a decimal number.

Test Case 9:
Operation: Division
Input 1: 8
Input 2: 0
Expected Output: Error or message "Cannot divide by zero"
Description: Handling division by zero error.

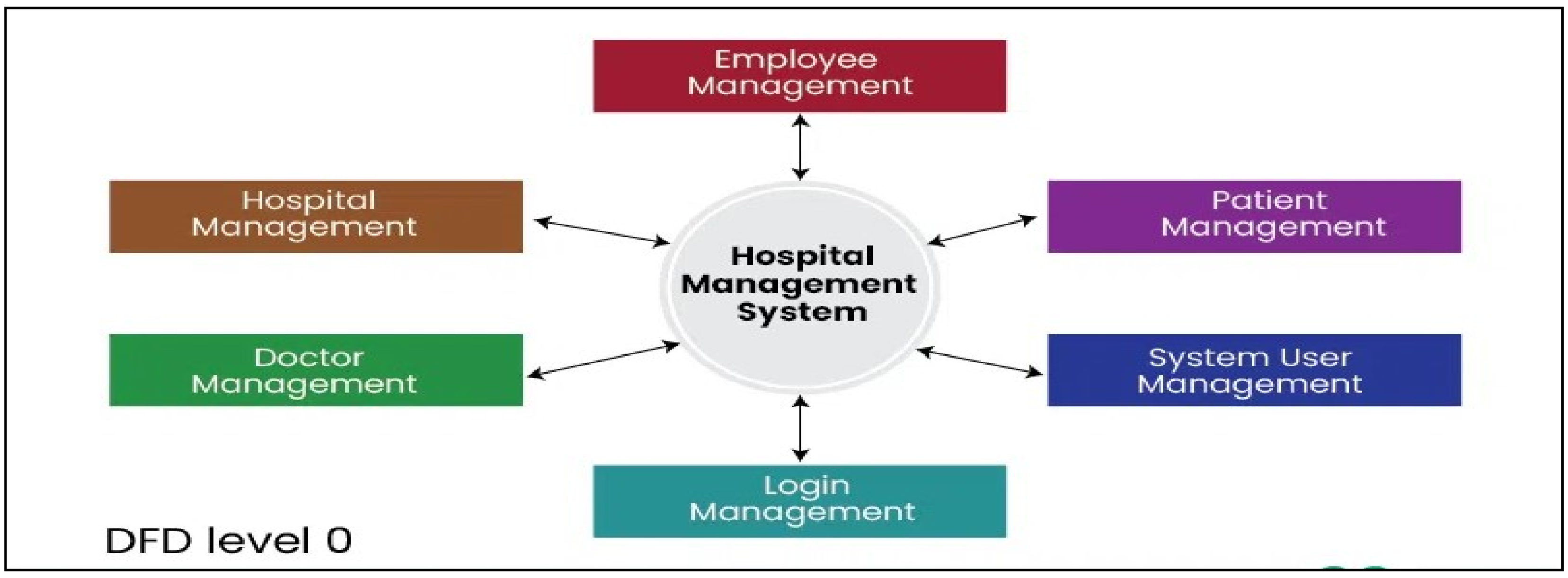
Test Case 10:
Operation: Addition
Input 1: 0
Input 2: 0
Expected Output: 0
Description: Adding zero to zero.

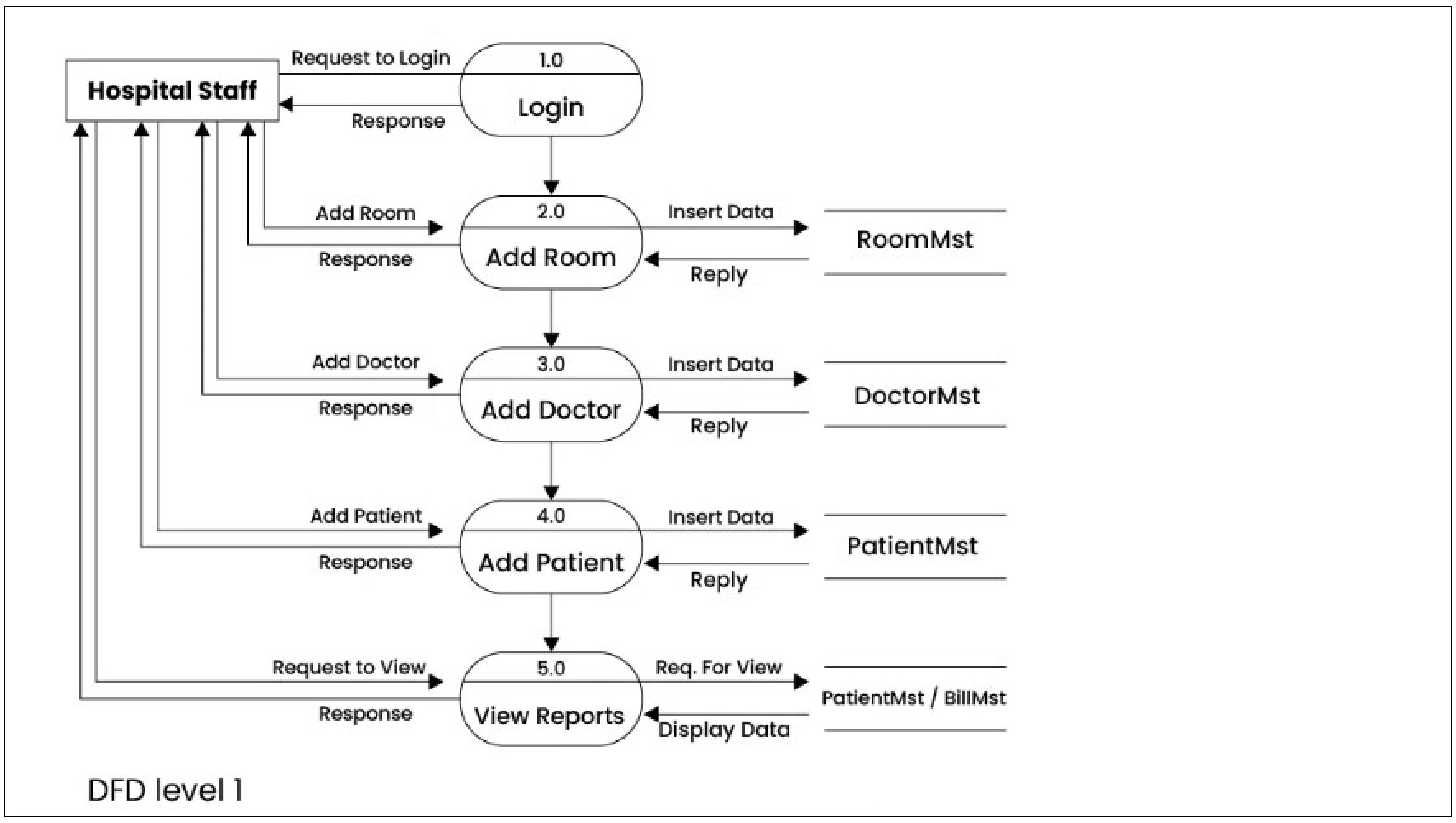
Test Case 11:
Operation: Subtraction
Input 1: 0
Input 2: 5
Expected Output: -5
Description: Subtracting from zero.

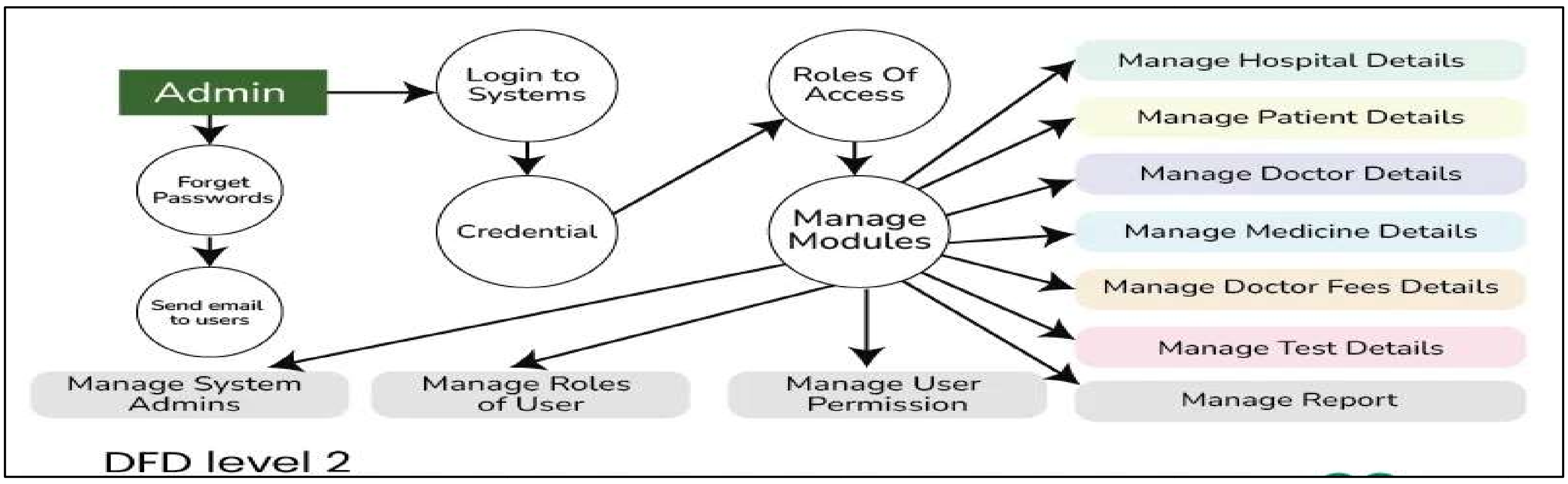
Test Case 12:
Operation: Multiplication
Input 1: 0
Input 2: 100
Expected Output: 0
Description: Multiplying any number by zero.

Test Case 13:
Operation: Division
Input 1: 0
Input 2: 10
Expected Output: 0
Description: Zero divided by a number.

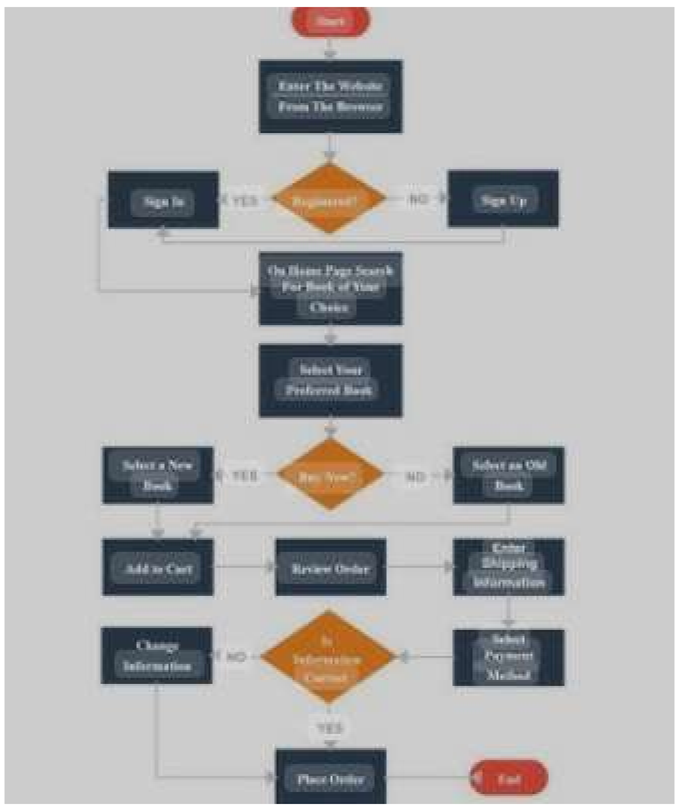
LAB EXERCISE: Create a DFD for a hospital management system







LAB EXERCISE: Draw a flowchart representing the logic of a basic online registration system.


Start
Enter The Website From The Browser
Registered?
YES
NO
Sign In
Sign Up
On Home Page Search For Book of Your Choice
Select Your Preferred Book
Buy New?
YES
NO
Select a New Book
Select an Old Book
Add to Cart
Review Order
Enter Shipping Information
Select Payment Method
Is Information Correct
NO
YES
Change Information
Place Order
End