

Module 1 – Overview of IT Industry

LAB EXERCISE

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

In C language

```
C #include <stdio.h> // Library for input/ output functions
1  #include <stdio.h> // Library for input/output functions
2
3  int main() {
4      printf("Hello World\n"); // Print to console
5      return 0;                // End program
6  }
7  |
```

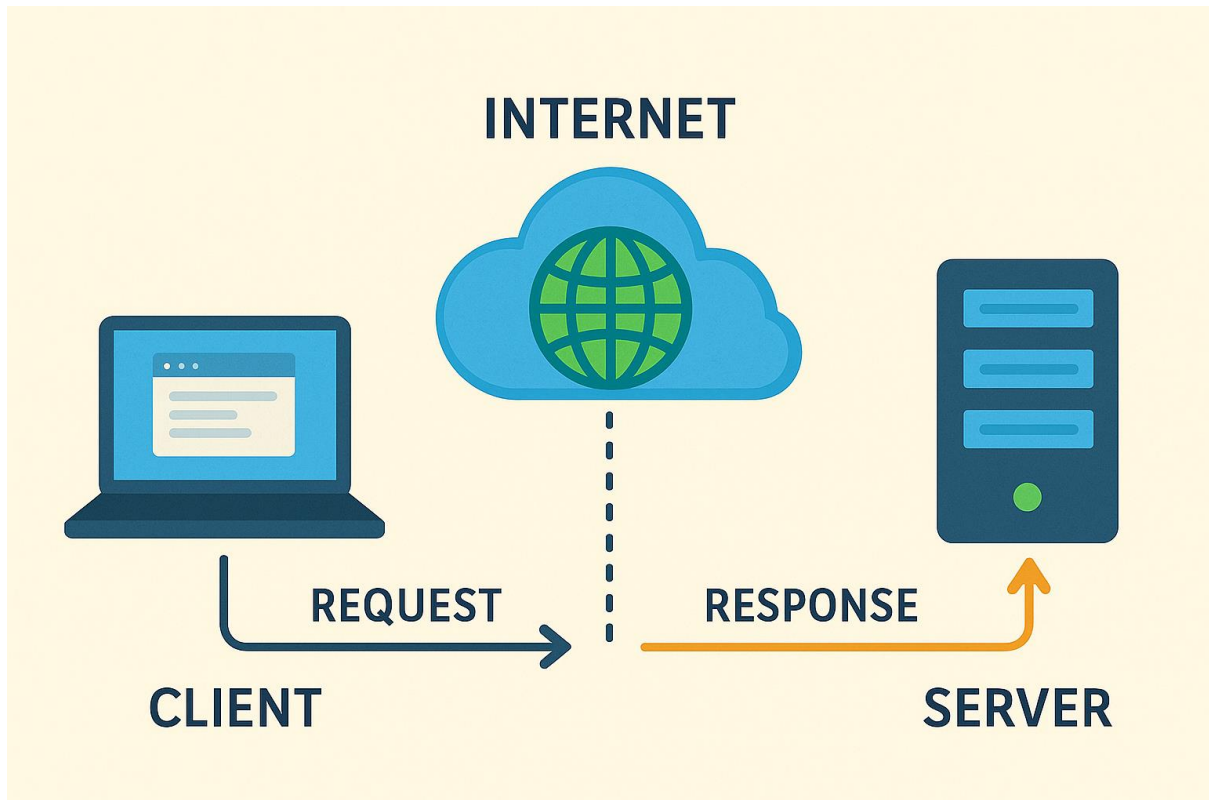
In python

```
C print("Hello World") Untitled-1 ●
1  print("Hello World")
2  |
```

Aspect	C	Python
Language Type	Compiled	Interpreted
Setup Required	Needs #include <stdio.h> for printing	No imports needed for basic printing
Main Function	Must define main() as the program entry point	No main function required
Semicolons	Required at the end of statements	Not used
Braces {}	Used to define code blocks	Uses indentation instead
Printing	Uses printf() with format specifiers	Uses print() with automatic formatting

Aspect	C	Python
Compilation	Must compile (e.g., gcc file.c -o file) before running	Runs directly (python file.py)

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



3. Design a simple HTTP client-server communication in any language.

Simple HTTP Server in C

```
File Edit Selection View Go Run Terminal Help Practical Assignments
C #include <stdio.h> Untitled-1
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 #define PORT 8080
8
9 int main() {
10     int server_fd, new_socket;
11     struct sockaddr_in address;
12     int addrlen = sizeof(address);
13     char buffer[30000] = {0};
14
15     const char *http_response =
16         "HTTP/1.1 200 OK\r\n"
17         "Content-Type: text/plain\r\n"
18         "Content-Length: 19\r\n"
19         "\r\n"
20         "Hello from server!";
21
22     // Create socket
23     server_fd = socket(AF_INET, SOCK_STREAM, 0);
24     if (server_fd == 0) {
25         perror("socket failed");
26         exit(EXIT_FAILURE);
27     }
28
29     // Bind socket to port
30     address.sin_family = AF_INET;
31     address.sin_addr.s_addr = INADDR_ANY;
32     address.sin_port = htons(PORT);
33
34     if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
35         perror("bind failed");
36         close(server_fd);
37         exit(EXIT_FAILURE);
38     }
39 }
```

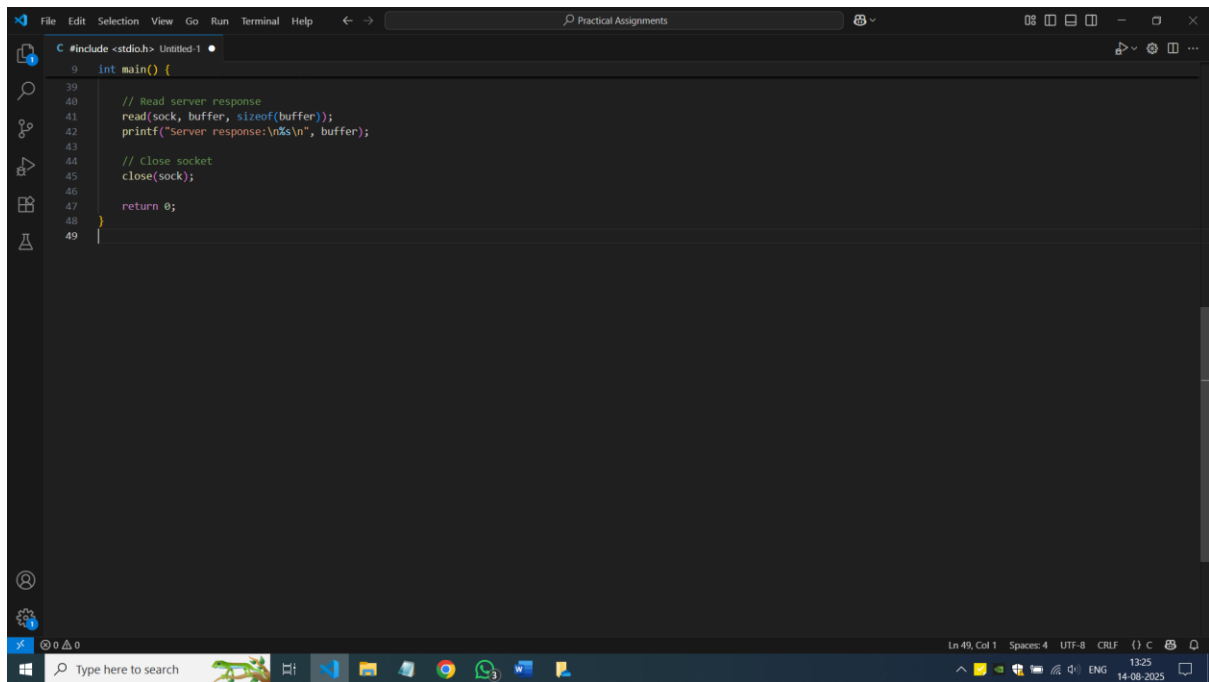
Ln 70, Col 1 Spaces: 4 UTF-8 CRLF {} C

Type here to search 31°C Haze 13:23 14-08-2025

```
File Edit Selection View Go Run Terminal Help Practical Assignments
C #include <stdio.h> Untitled-1
9 int main() {
39
40 // Listen for connections
41 if (listen(server_fd, 3) < 0) {
42     perror("listen");
43     close(server_fd);
44     exit(EXIT_FAILURE);
45 }
46 printf("Server running on port %d...\n", PORT);
47
48 // Accept one connection
49 new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen);
50 if (new_socket < 0) {
51     perror("accept");
52     close(server_fd);
53     exit(EXIT_FAILURE);
54 }
55
56 // Read request from client
57 read(new_socket, buffer, sizeof(buffer));
58 printf("Received request:\n%s\n", buffer);
59
60 // Send HTTP response
61 write(new_socket, http_response, strlen(http_response));
62 printf("Response sent.\n");
63
64 // Close sockets
65 close(new_socket);
66 close(server_fd);
67
68 return 0;
69 }
70
```

Simple HTTP Client in C

```
File Edit Selection View Go Run Terminal Help Practical Assignments
C #include <stdio.h> Untitled-1
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6
7 #define PORT 8080
8
9 int main() {
10     int sock = 0;
11     struct sockaddr_in serv_addr;
12     char buffer[30000] = {0};
13
14     // Create socket
15     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
16         printf("\n Socket creation error \n");
17         return -1;
18     }
19
20     serv_addr.sin_family = AF_INET;
21     serv_addr.sin_port = htons(PORT);
22
23     // Convert IPv4 and IPv6 addresses from text to binary
24     if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
25         printf("\nInvalid address/ Address not supported \n");
26         return -1;
27     }
28
29     // Connect to server
30     if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
31         printf("\nconnection Failed \n");
32         return -1;
33     }
34
35     // Send HTTP GET request
36     char *request = "GET / HTTP/1.1\r\nHost: localhost\r\n\r\n";
37     send(sock, request, strlen(request), 0);
38     printf("HTTP request sent.\n");
39 }
```



```
File Edit Selection View Go Run Terminal Help
C #include <stdio.h> Untitled-1
9 int main() {
39
40 // Read server response
41 read(sock, buffer, sizeof(buffer));
42 printf("Server response:\n%s\n", buffer);
43
44 // close socket
45 close(sock);
46
47 return 0;
48 }
49
```

Ln 49, Col 1 Spaces: 4 UTF-8 CRLF {} C 13:25 14-08-2025

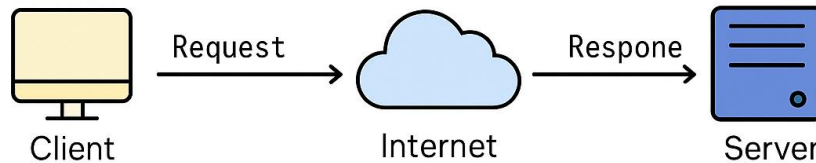
4. Research different types of internet connections (e.g., broadband, fiber, satellite) and list their pros and cons.

Internet Connections

Internet Type	Pros	Cons
Broadband (DSL / Cable)	Widely available in urban/suburban areas Affordable plans	DSL is slower than cable Speeds drop during peak times
Fiber-Optic	Extremely fast (up to gigabits) Low latency, ideal for gaming and streaming	Limited availability, especially in rural areas Higher installation cost
Satellite	Works in remote/rural areas No physical cables Can be accessed almost anywhere with a clear sky	High latency Slower than fiber or cable Weather can disrupt connection Often data caps
Mobile / Cellular (4G / 5G)	Portable Work anywhere with coverage Easy setup 5G offers speeds similar to fiber	Coverage varies Data plans can be expensive Speeds drop during congestion
Dial-Up	Very cheap Works with basic phone lines	Very slow (~56 kbps) Can't use phone and internet

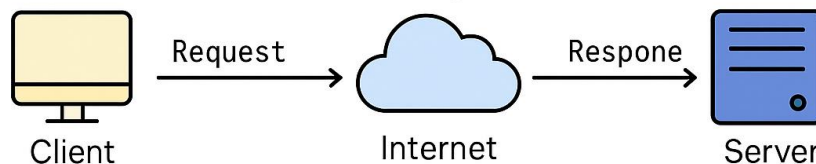
5. Simulate HTTP and FTP requests using command line tools (e.g., curl).

HTTP Requests



```
curl http://example.com
curl -X POST http://example.com/api -H
"Content-Type: application/json" -d '{"name': 'John', 'a25}'
```

FTP Requests



```
curl -T localfile.txt ftp://ftp.example.com/ --user username
password
curl ftp://ftp.example.com/ --user username:password
```

6. Identify and explain three common application security vulnerabilities. Suggest possible solutions.

1. SQL Injection (SQLi)

SQL injection is a code injection technique used to attack data-driven applications. An attacker inserts or "injects" a malicious SQL statement into an entry field for execution, such as a username field in a login form. If the application doesn't properly sanitize the input, the malicious code can be executed by the database. This can lead to unauthorized access, data theft, data modification, or even the deletion of the entire database.

Solution The most effective way to prevent SQL injection is to use **parameterized queries** (also known as prepared statements). These queries separate the SQL logic from the user-supplied data, ensuring that the input is treated as a value and not as an executable command. This prevents an attacker from altering the query's structure. Additionally, you

should implement the principle of least privilege by limiting the database user's permissions to only what is necessary for the application.

2. Cross-Site Scripting (XSS)

Cross-site scripting (XSS) is a type of injection attack where an attacker injects malicious scripts into a web application. The malicious script is then executed in the victim's browser when they visit the compromised web page. These scripts can steal sensitive information like cookies, session tokens, or other data stored by the browser. XSS attacks can be categorized into three main types:

- **Stored XSS:** The malicious script is permanently stored on the target server (e.g., in a database) and is delivered to users who visit the web page.
- **Reflected XSS:** The malicious script is reflected off the web server and is executed in the user's browser, typically via a malicious link or a form.
- **DOM-based XSS:** The vulnerability lies in the client-side code itself, not the server-side, and the attack is executed by manipulating the page's Document Object Model (DOM).

Solution To prevent XSS, you must **sanitize and encode user input** on the server side before it is displayed on a web page. This involves converting special characters, such as <, >, and &, into their HTML entities (<, >, &), so they are rendered as text rather than being executed as code. Another crucial defense is implementing a **Content Security Policy (CSP)**, which acts as a whitelist to restrict the sources from which scripts can be loaded and executed.

3. Broken Authentication and Session Management

This vulnerability occurs when an application's authentication and session management functions are improperly implemented, allowing an attacker to gain unauthorized access to user accounts. This can be due to weak

password policies, insecure storage of credentials, predictable session IDs, or poor handling of session timeouts. Attackers can exploit these flaws through brute-force attacks, credential stuffing, or by stealing a user's session token to impersonate them.

Solution You can fix this by implementing **strong authentication mechanisms**. This includes using a strong password policy, hashing and salting passwords securely, and implementing multi-factor authentication (MFA). For session management, use secure, randomly generated session IDs and ensure that they are transmitted only over a secure connection (HTTPS). It's also important to implement proper session expiration and to invalidate sessions upon logout or after a period of inactivity.

7. Identify and classify 5 applications you use daily as either system software or application software.

Application Software

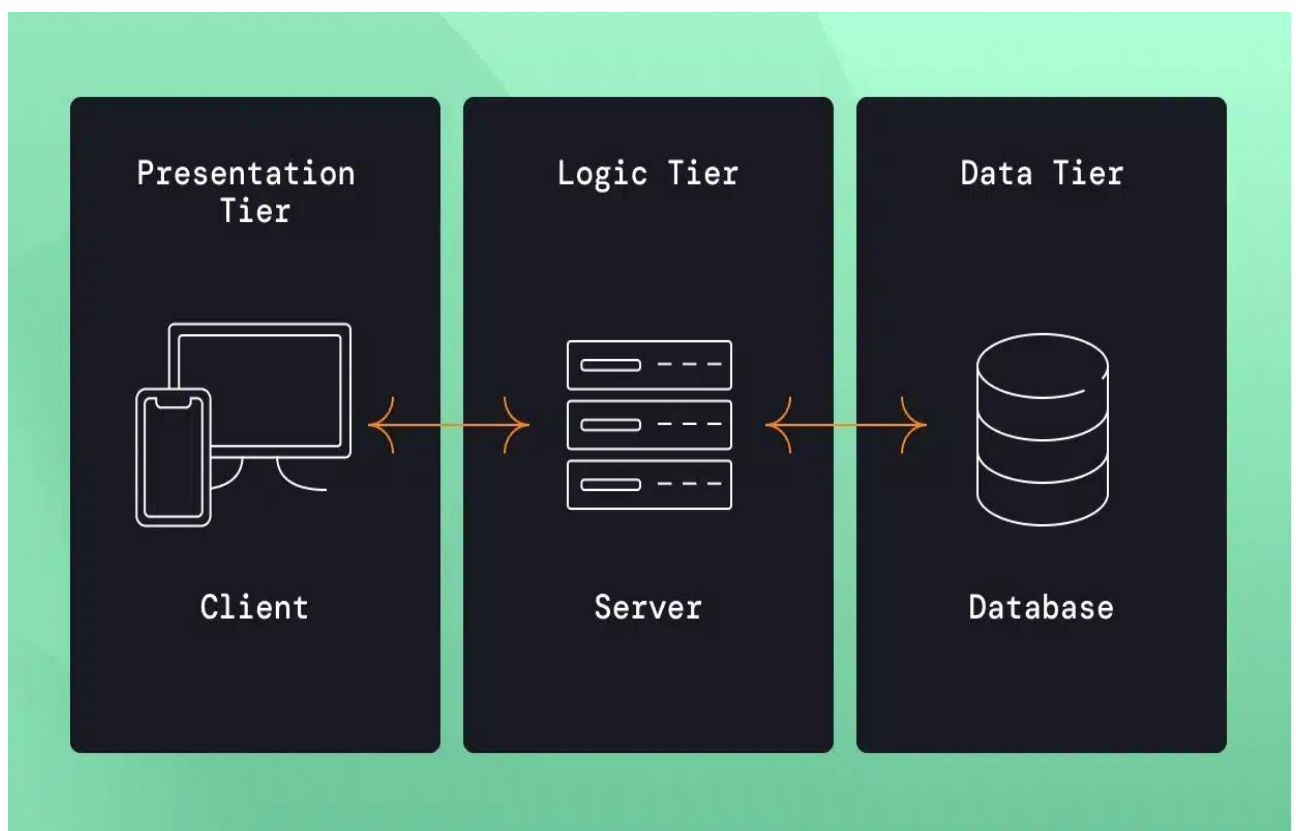
- **Google Chrome:** A web browser that allows you to access and view websites. It performs a specific task for the end-user, which is web Browse.
- **Microsoft Word:** A word processing program used to create, edit, and format documents. It's designed to perform a specific function for the user.
- **Microsoft Excel:** A spreadsheet application used for data entry, calculations, and analysis. Its purpose is to assist the user with a specific task.

System Software

- **macOS:** This is the operating system for Apple's Mac computers. An operating system is a fundamental piece of system software that manages a computer's hardware and software resources, providing a platform for other programs (application software) to run on.

- **iOS:** The mobile operating system that powers Apple's iPhone and iPad devices. Like macOS, it manages the device's resources and provides the core environment for all applications to function.

8. Design a basic three-tier software architecture diagram for a web application.



9. Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.

Case Study: Three-Tier Architecture in an Online Food Delivery Application

Overview

This case study focuses on an online food delivery platform (similar to Swiggy or Uber Eats) that follows the **three-tier architecture** model:

1. **Presentation Layer (Client/UI Layer)**
 2. **Business Logic Layer (Application Layer)**
 3. **Data Access Layer (Database Layer)**
-

1. Presentation Layer

Functionality:

- Acts as the **user interface** of the system.
- Displays restaurant lists, menus, cart, and order tracking.
- Captures user input such as login credentials, food selections, and payment details.

Key Components:

- Mobile app (iOS/Android) or web frontend.
- Implemented using HTML, CSS, JavaScript (React or Angular), and mobile frameworks like Flutter.

Example Actions:

- User searches for “Pizza.”
 - The app displays a filtered list of restaurants offering pizza.
 - The user adds a pizza to the cart and proceeds to checkout.
-

2. Business Logic Layer

Functionality:

- Handles the **core processing** of the application.

- Validates user credentials, processes orders, calculates discounts, and manages payment workflows.
- Implements rules such as “Free delivery on orders above \$20.”

Key Components:

- Written in backend frameworks like **Node.js, Java Spring Boot, or Django**.
- Communicates with APIs and manages sessions.

Example Actions:

- Verifies that the restaurant is open before accepting the order.
 - Applies coupon codes and calculates the final bill.
 - Sends the order details to the kitchen system.
-

3. Data Access Layer

Functionality:

- Manages the **storage and retrieval** of data from databases.
- Keeps all restaurant menus, customer details, and order histories safe.
- Ensures secure transactions and data integrity.

Key Components:

- Databases such as **MySQL, PostgreSQL, or MongoDB**.
- ORM (Object-Relational Mapping) tools like Hibernate or Sequelize.

Example Actions:

- Retrieves the pizza menu from the database when requested.
 - Stores the user’s order history.
 - Updates order status from “Cooking” to “Out for Delivery.”
-

Workflow Example

1. **User Action (Presentation Layer):**
User clicks “Place Order” on the app.
2. **Processing (Business Logic Layer):**
Validates the order, processes payment, and confirms restaurant availability.
3. **Data Handling (Data Access Layer):**
Saves order details in the database and retrieves estimated delivery time.

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

11. Write and upload your first source code file to Github.

<https://github.com/Kirito-420/module1-.git>

12. a Github repository and document how to commit and push code changes

1. Create a GitHub Repository

1. Log in to [GitHub](#).
2. Click the “+” → **New repository**.
3. Give it a name (e.g., my-project), choose **Public** or **Private**.
4. Check **Add a README** if you want a starter file.
5. Click **Create repository**.

2.Clone the Repository to Your Local Machine

```
MINGW64/c/Users/Jiniyas Suthar
jiniyas_suthar@DESKTOP-INJHNG7 MINGW64 ~
$ git clone https://github.com/kirito-420/practice-.git
Cloning into 'practice'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
jiniyas_suthar@DESKTOP-INJHNG7 MINGW64 ~
$
```

3. Make Changes Locally

```
MINGW64/c/Users/Jiniyas Suthar
jiniyas_suthar@DESKTOP-INJHNG7 MINGW64 ~
$ git clone https://github.com/kirito-420/practice-.git
Cloning into 'practice'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
jiniyas_suthar@DESKTOP-INJHNG7 MINGW64 ~
$ echo "Hello, Github!" > hello.txt
jiniyas_suthar@DESKTOP-INJHNG7 MINGW64 ~
$
```

4. Check Repository Status

```
MINGW64/c/Users/Jiniyas Suthar
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ git clone https://github.com/Kirito-420/practice-.git
Cloning into 'practice'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ echo "Hello, Github!" > hello.txt
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ git status
fatal: not a git repository (or any of the parent directories): .git
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$
```

5. Stage the Changes

```
MINGW64/c/Users/Jiniyas Suthar
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ git clone https://github.com/Kirito-420/practice-.git
Cloning into 'practice'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ echo "Hello, Github!" > hello.txt
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ git status
fatal: not a git repository (or any of the parent directories): .git
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ git add hello.txt
fatal: not a git repository (or any of the parent directories): .git
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ echo "Hello, Github!"
Hello, Github!
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ git status
fatal: not a git repository (or any of the parent directories): .git
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$ git add hello.txt
fatal: not a git repository (or any of the parent directories): .git
jiniyas.Suthar@DESKTOP-INJH67 MINGW64 ~
$
```

6. Commit the Changes


```
MINGW64/c/Users/Jiniyas Suthar
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git clone https://github.com/Kirito-420/practice-.git
Cloning into 'practice-'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ echo "Hello, Github!" > hello.txt
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git status
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git add hello.txt
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ echo "Hello, Github!"
Hello, Github!
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git status
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git add hello.txt
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git commit -m "Add hello.txt with greeting"
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$
```

7. Push Changes to GitHub

```
MINGW64/c/Users/Jiniyas Suthar
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git clone https://github.com/Kirito-420/practice-.git
Cloning into 'practice-'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ echo "Hello, Github!" > hello.txt
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git status
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git add hello.txt
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ echo "Hello, Github!"
Hello, Github!
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git status
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git add hello.txt
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git commit -m "Add hello.txt with greeting"
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$ git push origin main
fatal: not a git repository (or any of the parent directories): .git
Nityas_Suthar@DESKTOP-INJHG7 MINGW64 ~
$
```

8. Verify on GitHub

- Go to your repository on GitHub.
- Refresh the page — you'll see hello.txt added.

13.: Create a student account on Github and collaborate on a small project with a classmate.

Step 1: Create a Student GitHub Account

- 1. Go to GitHub Education.**
 - 2. Click “Get benefits”.**
 - 3. Sign up (or log in) with your email (preferably your school/college email).**
 - 4. Fill in the application:**
 - **Upload proof of student status (student ID or enrollment letter).**
 - **Provide your expected graduation year.**
 - 5. Once approved, you’ll get GitHub Pro for free (extra features + free private repos).**
-

Step 2: Set Up a Practice Project

- 1. One student creates a new repository:**
 - **Go to GitHub → New Repository.**
 - **Name it: student-collab-project.**
 - **Initialize with a README.md.**
- 2. Add a collaborator (classmate):**
 - **Open the repo → Settings → Collaborators.**

- Invite your classmate by their GitHub username or email.
 - They'll receive an email invite.
-

Step 3: Clone the Repository

Both students run:

```
git clone https://github.com/YOUR-  
USERNAME/student-collab-project.git  
cd student-collab-project
```

Step 4: Create Branches for Work

- **Student A creates a branch for frontend:**
 - `git checkout -b frontend`
- **Student B creates a branch for backend:**
 - `git checkout -b backend`

Both students work separately without touching main.

Step 5: Make Changes

- **Student A edits frontend.txt:**
 - This is the frontend part of the project.
 - `git add frontend.txt`

- **git commit -m "Add frontend work"**
 - **git push origin frontend**
 - **Student B edits backend.txt:**
 - **This is the backend part of the project.**
 - **git add backend.txt**
 - **git commit -m "Add backend work"**
 - **git push origin backend**
-

Step 6: Merge Using Pull Requests

- 1. Go to the GitHub repository online.**
 - 2. GitHub will suggest:**
 - **“Compare & pull request” for frontend.**
 - **“Compare & pull request” for backend.**
 - 3. Each student creates a Pull Request (PR).**
 - 4. The other student reviews the PR and clicks Merge.**
 - 5. Both contributions are now merged into main.**
- **Both students can now pull the updated main:**
 - **git checkout main**

git pull origin main

14. Create a list of software you use regularly and classify them into the following categories: system, application, and utility software.

System Software		
Windows 11 Operating system that manages hardware and software resources	Linux (Ubuntu) Open-source OS for managing system operations	
Application Software		
Google Chrome Web browser for accessing the internet	Microsoft Word Word processor for creating and editing documents	
WhatsApp Messaging and calling application	VLC Media Player Multimedia player for audio and video files	
Utility Software		
WinRAR File compression and extraction tool	Antivirus Software (e.g. Avast) Protects the system from malware and viruses	Disk Management Helps manage hard drive partitions and storage

15. Follow a GIT tutorial to practice cloning, branching, and merging repositories

1. cloning

```
MINGW64/f/git
jinyas Suthar@DESKTOP-INJHG7 MINGW64 ~
$ cd "f:\git"

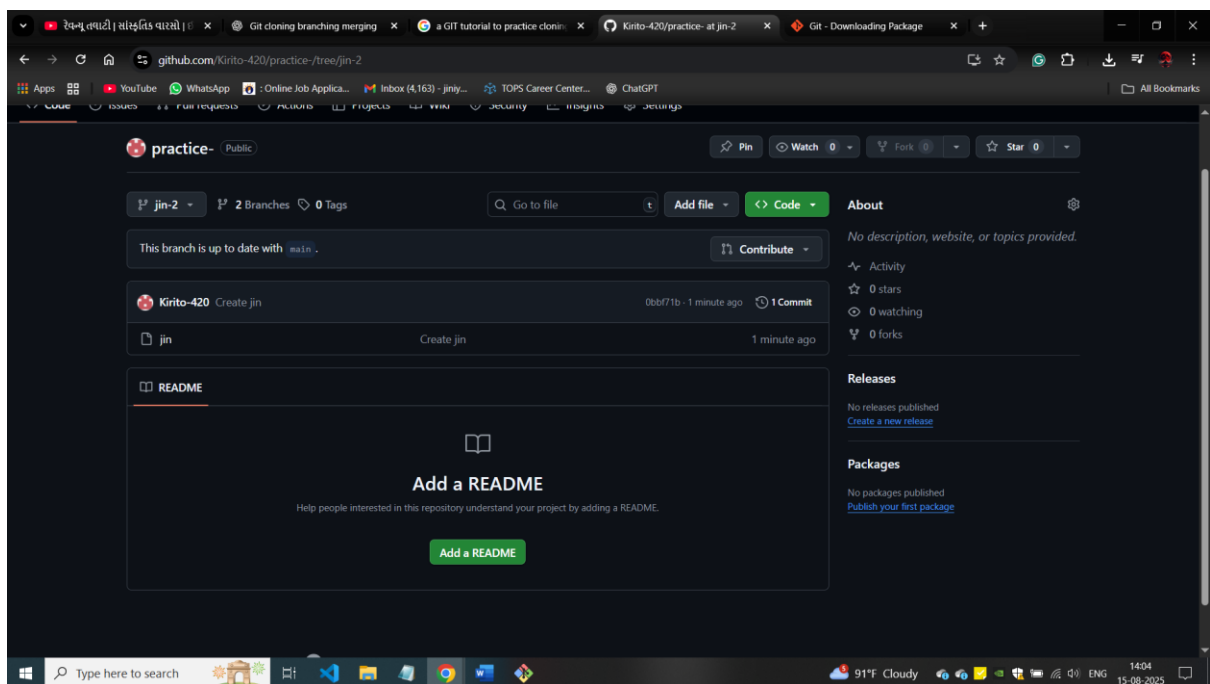
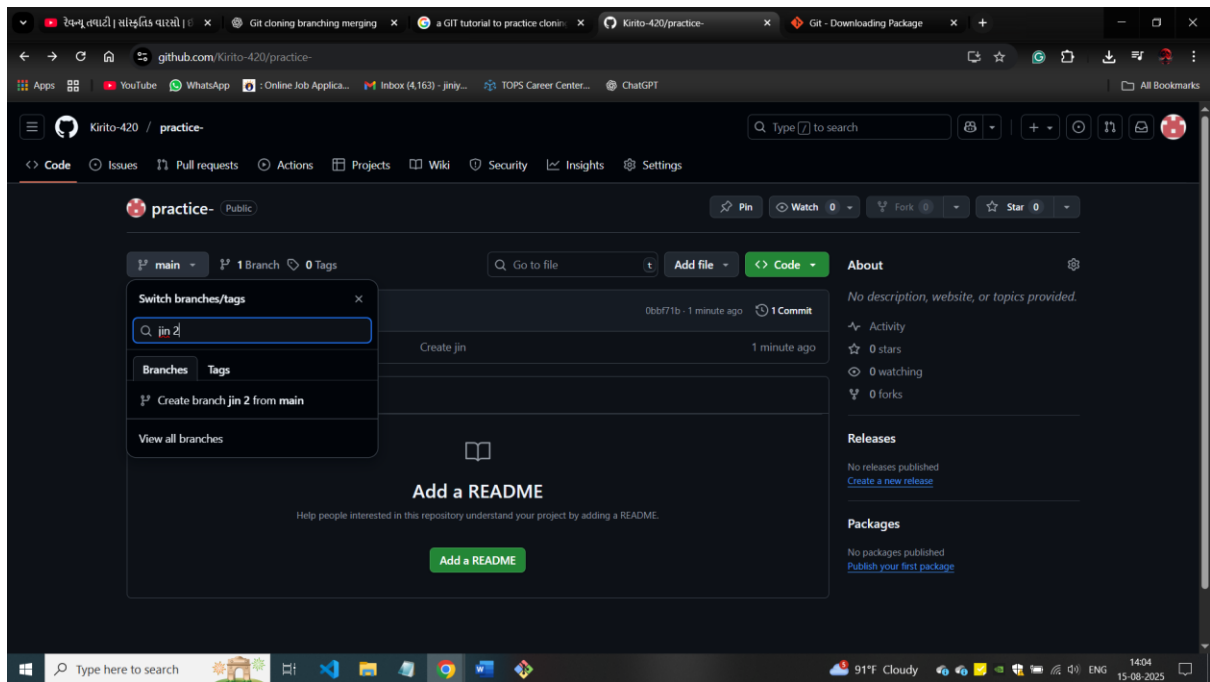
jinyas Suthar@DESKTOP-INJHG7 MINGW64 /f/git
$ git clone "https://github.com/Kirito-420/module1-.git"
Cloning into 'module1-...'
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), 323.70 KiB | 1.09 MiB/s, done.

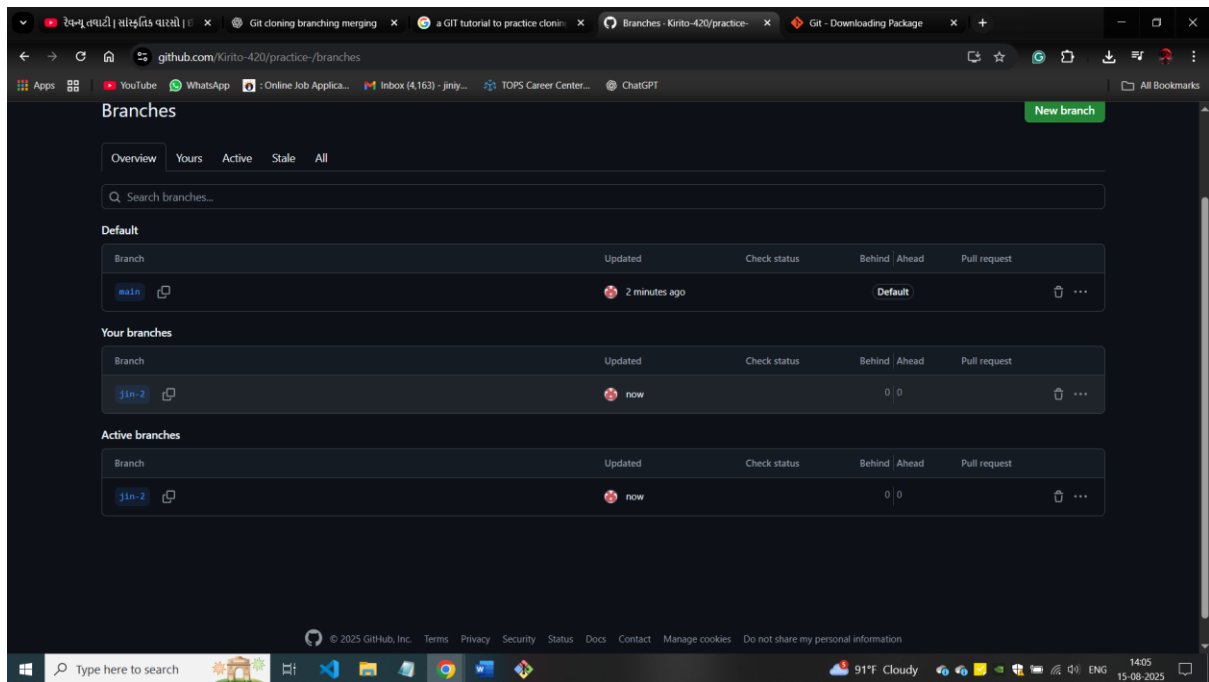
jinyas Suthar@DESKTOP-INJHG7 MINGW64 /f/git
$ ls
module1-/

jinyas Suthar@DESKTOP-INJHG7 MINGW64 /f/git
$ |
```

2.Branching

The screenshot shows a web browser displaying the GitHub repository page for 'practice-'. The repository is public and has 0 stars, 0 forks, and 0 releases. A 'Switch branches/tags' dialog is open, showing the 'main' branch as the current branch. The dialog also includes a search bar for finding or creating a branch, and a list of branches (main) and tags (0). The repository page itself shows a 'Go to file' search bar, an 'Add file' button, and a 'Code' button. The 'About' section on the right indicates that there is no description, website, or topics provided for this repository.





3.Merging Repositories

Open your terminal and run:

```
git clone https://github.com/YOUR-USERNAME/practice-git.git  
cd practice-git
```

Step 3: Create a New Branch

Let's say we want to create a feature branch.

```
git checkout -b feature
```

Edit the README.md (or create a new file feature.txt) and add:

This is my feature branch work.

Save the file and commit:

```
git add .
```

```
git commit -m "Add feature branch work"
```

Push it to GitHub:

git push origin feature

Step 4: Merge Branch into Main

Now let's merge the branch back into main.

- 1. Switch to main:**
 - 2. git checkout main**
 - 3. Merge feature branch:**
 - 4. git merge feature**
 - 5. Push changes:**
 - 6. git push origin main**
-

Step 5: Try Conflicts (Optional but Fun 😄)

Let's create a conflict so you learn to resolve it.

- 1. On main, open README.md and add:**
- 2. Line from main branch**

Commit and push.

- 3. Switch to feature branch and add:**
- 4. Line from feature branch**

Commit and push.

- 5. Now try merging feature into main → Git will show a merge conflict.**
- 6. Open the conflicted file. You'll see:**
- 7. <<<<<< HEAD**
- 8. Line from main branch**
- 9. =====**

10. **Line from feature branch**

11. **>>>>>> feature**

Edit it to keep what you want, then:

git add README.md

git commit -m "Resolved conflict"

git push origin main.

16. Write a report on the various types of application software and how they improve productivity.

1. Introduction

Application software refers to programs designed to perform specific tasks for the end user. Unlike system software, which runs the hardware and system operations, application software is focused on helping users accomplish activities such as creating documents, managing data, editing media, and communicating. The right application software can greatly improve productivity by automating repetitive tasks, enabling collaboration, and providing specialized tools.

2. Types of Application Software

2.1 Productivity Software

- **xamples:** Microsoft Office (Word, Excel, PowerPoint), Google Workspace (Docs, Sheets, Slides), LibreOffice.
- **Purpose:** Helps in creating documents, spreadsheets, and presentations.
- **Productivity Benefits:**
 - Streamlines content creation and formatting.

- Automates calculations and data analysis.
 - Enables sharing and real-time collaboration.
-

2.2 Database Management Software (DBMS)

- **Examples:** MySQL, Microsoft Access, Oracle Database.
 - **Purpose:** Organizes, stores, and retrieves data efficiently.
 - **Productivity Benefits:**
 - Reduces time spent searching for information.
 - Allows multiple users to work on the same dataset.
 - Improves decision-making through quick data access.
-

2.3 Communication Software

- **Examples:** Microsoft Teams, Zoom, Slack, WhatsApp.
 - **Purpose:** Facilitates messaging, voice calls, and video conferencing.
 - **Productivity Benefits:**
 - Reduces the need for physical meetings.
 - Enables instant sharing of files and updates.
 - Improves team coordination.
-

2.4 Graphic Design and Multimedia Software

- **Examples:** Adobe Photoshop, Canva, CorelDRAW, Final Cut Pro.
- **Purpose:** Creates and edits visual and audio content.
- **Productivity Benefits:**
 - Speeds up creative workflows.
 - Provides templates and automation tools for design.

- Improves quality and consistency of visual content.
-

2.5 Project Management Software

- **Examples:** Trello, Asana, Jira, Microsoft Project.
 - **Purpose:** Tracks tasks, deadlines, and project progress.
 - **Productivity Benefits:**
 - Improves organization and accountability.
 - Prevents missed deadlines.
 - Facilitates team coordination across locations.
-

2.6 Enterprise Software

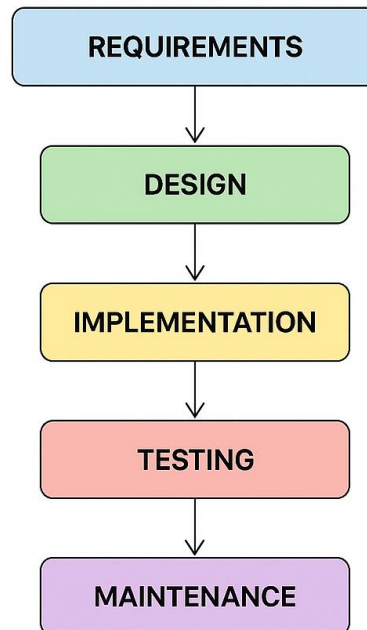
- **Examples:** SAP, Oracle ERP, Salesforce.
 - **Purpose:** Manages large-scale business operations.
 - **Productivity Benefits:**
 - Integrates various business functions (HR, finance, supply chain).
 - Provides data-driven decision-making tools.
 - Improves operational efficiency.
-

3. How Application Software Improves Productivity

1. **Automation:** Reduces manual effort (e.g., Excel formulas).
2. **Collaboration:** Enables teamwork in real-time.
3. **Accessibility:** Cloud-based apps allow work from anywhere.
4. **Accuracy:** Minimizes human errors in data and calculations.
5. **Time Management:** Helps track progress and deadlines.

17. Create a flowchart representing the Software Development Life Cycle (SDLC).

SOFTWARE DEVELOPMENT LIFE CYCLE



18. Write a requirement specification for a simple library management system.

1. Introduction

1.1 Purpose

The purpose of this Library Management System (LMS) is to automate and simplify the process of managing library resources, including books, members, and borrowing/returning activities. The system aims to reduce manual work, improve accuracy, and provide an easy-to-use interface for both library staff and members.

1.2 Scope

The LMS will manage:

- Book inventory (add, update, delete books).
- Member registration and management.

- Issuing and returning books.
- Fine calculation for late returns.
- Search functionality for books and members.

The system will be accessible via a desktop interface and will store all data in a database.

2. Functional Requirements

2.1 User Roles

1. Librarian

- Add, edit, or remove books.
- Register new members.
- Issue and return books.
- View reports (issued books, overdue books, fines).

2. Member

- Search for books.
 - View personal issued book history.
 - Request book renewal.
-

2.2 Features

1. Book Management

- Add new books with details (title, author, ISBN, category, quantity).
- Update book information.
- Delete books from the system.

2. Member Management

- Register new members with personal details.

- Update or remove member records.

3. Issuing/Returning Books

- Record issue date and return date.
- Automatically calculate due dates.
- Fine calculation for late returns.

4. Search Function

- Search by title, author, or ISBN.

5. Reports

- Generate lists of issued books, overdue books, and fines.
-

3. Non-Functional Requirements

1. Usability

- User-friendly interface with clear navigation.

2. Performance

- System should process operations within 2 seconds.

3. Security

- Role-based access (members cannot modify library records).
- Password protection for librarian accounts.

4. Data Integrity

- No duplicate book or member records.

5. Scalability

- Ability to handle up to 10,000 book records and 1,000 members.
-

4. System Requirements

- **Hardware:**
 - Processor: Intel i3 or above
 - RAM: 4GB or higher
 - Storage: 500MB free space
 - **Software:**
 - OS: Windows / Linux
 - Database: MySQL / SQLite
 - Language: Java / C# / Python
 - UI Framework: JavaFX, Tkinter, or .NET Forms
-

5. Constraints

- Offline desktop application (no cloud integration in first version).
 - Single-librarian login per session.
-

6. Assumptions

- All books have unique ISBN numbers.
- Members have unique ID numbers.
- Library operates within standard working hours for issuing/returning books.

19. Perform a functional analysis for an online shopping system.

Functional Analysis – Online Shopping System

1. Introduction

An Online Shopping System allows customers to browse, search, select, and purchase products over the internet. It connects buyers and sellers via a web-based interface and supports order management, payment processing, and delivery tracking.

2. Main Functional Areas

2.1 User Management

- **User Registration & Login**
 - Create new accounts with personal details.
 - Secure authentication (username/password, 2FA optional).
 - **Profile Management**
 - Update personal details, addresses, and payment preferences.
 - **Role-based Access**
 - Customers: Browse and purchase products.
 - Admins: Manage inventory, users, and orders.
-

2.2 Product Management

- **Add/Update/Delete Products** (Admin only)
 - Product details: name, category, description, price, images, stock quantity.
 - **Product Categorization**
 - Organize products into categories and subcategories.
 - **Search and Filter**
 - Search by name, category, price range, brand, etc.
-

2.3 Shopping Cart & Wishlist

- **Add to Cart**
 - Add products to a virtual cart for later purchase.
 - **Modify Cart**
 - Update quantity or remove items.
 - **Wishlist**
 - Save products for future reference.
-

2.4 Order Processing

- **Checkout**
 - Select delivery address, payment method, and confirm order.
 - **Order Tracking**
 - View order status: pending, shipped, delivered.
 - **Invoice Generation**
 - Generate digital receipts for orders.
-

2.5 Payment Processing

- **Multiple Payment Options**
 - Credit/Debit card, UPI, Net banking, COD.
 - **Secure Transactions**
 - Encryption (SSL), Payment Gateway integration.
 - **Refunds/Returns**
 - Initiate returns and process refunds.
-

2.6 Customer Support

- **Live Chat / Ticketing System**

- Allow customers to ask questions or report issues.
 - **FAQs & Help Center**
 - Provide self-service support.
-

3. Non-Functional Requirements

- **Performance:** Load pages within 3 seconds under normal traffic.
 - **Security:** Protect sensitive data with encryption and secure authentication.
 - **Scalability:** Handle increased users and product data during sales or festivals.
 - **Usability:** Intuitive UI for easy navigation.
 - **Availability:** 99.9% uptime.
-

4. Key Benefits

- Convenience: Shop anytime, anywhere.
- Wider Reach: Sellers can reach global customers.
- Efficiency: Faster product search and purchase process.
- Personalization: Recommendations based on purchase history.

20.: Design a basic system architecture for a food delivery app.

1. Overview

The food delivery app connects **customers**, **restaurants**, and **delivery partners** through a central platform that handles **ordering**, **payment**, and **delivery tracking**.

2. Components

A. Presentation Layer (Frontend)

- **Customer App / Website**
 - Browse restaurants & menus
 - Place orders & track delivery
 - Manage profile, addresses, and payment methods
- **Restaurant Dashboard**
 - Manage menus, prices, and availability
 - Accept or reject orders
- **Delivery Partner App**
 - Receive delivery requests
 - Navigate to pickup & drop-off locations

B. Application Layer (Backend Services)

- **Order Management Service**
 - Processes incoming orders from customers
 - Sends orders to restaurant systems
- **User Management Service**
 - Handles registration, login, and authentication
- **Menu Management Service**
 - Stores menu data, categories, prices, and images
- **Delivery Management Service**
 - Assigns delivery partners and manages real-time location tracking

- **Payment Service**
 - Integrates with payment gateways for secure transactions
 - **Notification Service**
 - Sends order updates via push notifications, SMS, or email
-

C. Data Layer

- **Databases**
 - **User Database** (customers, restaurants, delivery partners)
 - **Menu Database** (items, categories, prices)
 - **Order Database** (order history, statuses)
 - **Payment Records** (transactions, refunds)
 - **Cache Layer**
 - Redis/Memcached for quick menu and location lookups
-

D. External Integrations

- **Payment Gateway** (Razorpay, Stripe, PayPal)
 - **Map & Navigation API** (Google Maps API)
 - **SMS/Email API** for alerts
-

3. Flow

1. **Customer places an order** via the app → **Order Service** processes it.
2. **Order sent to the restaurant** → restaurant accepts & starts preparing food.
3. **Delivery Service assigns** a nearby delivery partner → partner picks up order.
4. **Payment Service** confirms transaction.

5. **Customer receives real-time tracking** until delivery is complete.

21. Document a real-world case where a software application required critical maintenance.

Case Study: Boeing 737 MAX MCAS System Update

1. Background

The **Boeing 737 MAX** aircraft was grounded worldwide in March 2019 after **two fatal crashes** (Lion Air Flight 610 and Ethiopian Airlines Flight 302) that killed 346 people.

Investigations revealed a **software issue** in the **Maneuvering Characteristics Augmentation System (MCAS)** — an automated flight control system.

2. The Problem

- MCAS was designed to automatically push the plane's nose down if it detected the aircraft was climbing too steeply.
 - The system **relied on a single Angle of Attack (AoA) sensor**.
 - If that sensor failed or gave incorrect data, MCAS could activate repeatedly — even when not needed — forcing the plane's nose down.
 - Pilots were not fully trained or informed about MCAS behavior, making it difficult to counteract during emergencies.
-

3. Type of Maintenance

Corrective Maintenance — fixing a critical bug that directly caused safety issues.

4. The Maintenance Process

- **Software redesign** to take data from **both AoA sensors** instead of one.
- **Fail-safe logic** added: MCAS would deactivate if sensors disagreed.
- **Reduced activation authority** so pilots could manually override MCAS more easily.
- **Pilot training program** updated to include MCAS operation and emergency response procedures.
- **Extensive testing and simulation** before aircraft re-certification by the FAA.

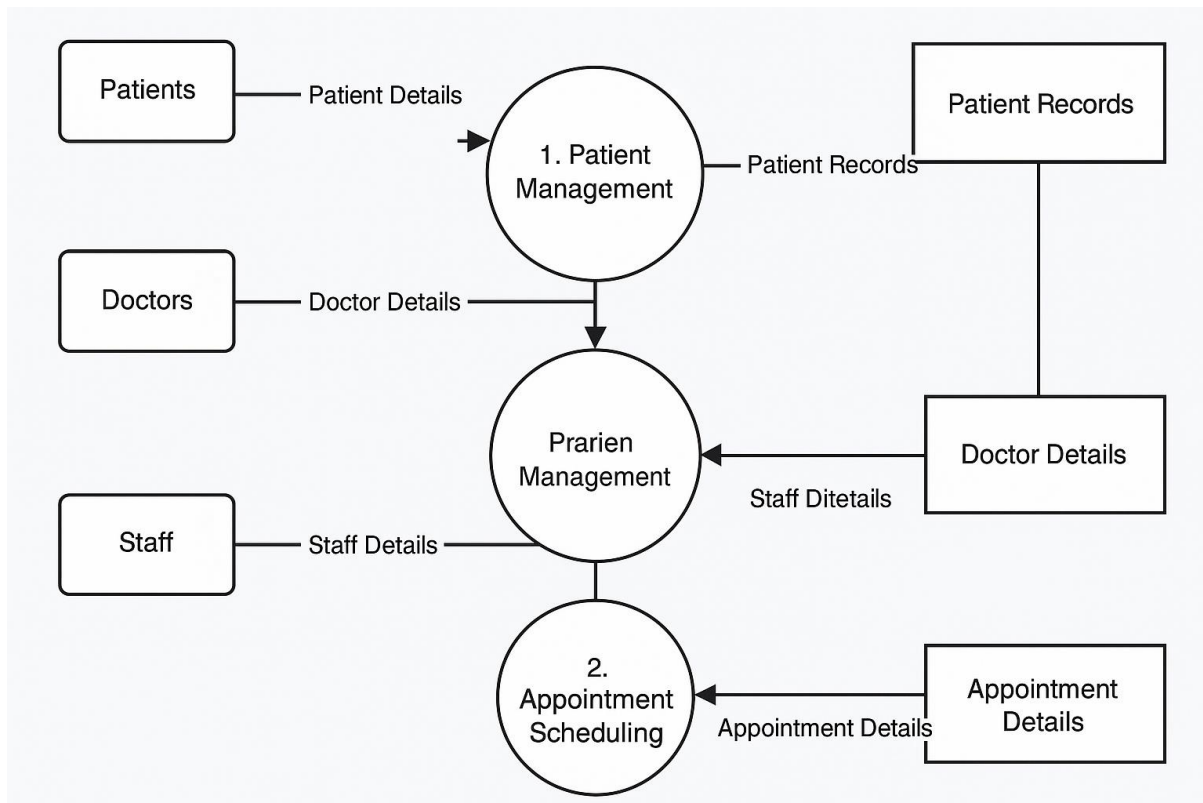
5. Outcome

- In November 2020, the **FAA cleared the Boeing 737 MAX** to return to service after approving the software changes.
- The incident became one of the most significant examples of how **critical software maintenance** can directly affect **human safety**.

6. Lessons Learned

- Always include **redundancy** for critical safety systems.
- **Transparency and training** for end-users (pilots, operators) are vital.
- Rigorous **testing under failure conditions** can prevent catastrophic consequences.

22. Create a DFD for a hospital management system



23. Build a simple desktop calculator application using a GUI library.

```

import tkinter as tk
import tkinter as tk

def click(event):
    global expression
    text = event.widget.get("text")
    if text == "=":
        try:
            result = str(eval(expression))
            entry_var.set(result)
            expression = result
        except Exception:
            entry_var.set("Error")
            expression = ""
    elif text == "C":
        expression = ""
        entry_var.set("")
    else:
        expression += text
        entry_var.set(expression)

# Main window
root = tk.Tk()
root.title("Simple Calculator")
root.geometry("300x400")

expression = ""
entry_var = tk.StringVar()

# Display
entry = tk.Entry(root, textvar=entry_var, font="Arial 20", justify="right")
entry.pack(fill="both", ipadx=8, pady=10)

# Buttons layout
buttons = [
    ["7", "8", "9", "."],
    ["4", "5", "6", "-"],
    ["1", "2", "3", "+"],
    ["C", "=", "X", "/"]
]

for (row, col), button in enumerate(buttons):
    for text in button:
        tk.Button(root, text=text, width=50, height=30, command=lambda: click(event)).pack(side="top", fill="both", ipadx=5, ipady=5)
  
```

The screenshot shows a code editor with Python code for a simple desktop calculator application. The code uses the Tkinter GUI library. It defines a `click` function that handles button presses, updating the expression and the display. The main window is titled "Simple Calculator" and has a geometry of 300x400. The display is a Tk.Entry widget with a font of Arial 20, justified right. The buttons are arranged in a grid, with each button having a width of 50 and a height of 30. The buttons are labeled with digits, operators, and a clear button (C). The code is written in Python and uses the Tkinter library.


```
File Edit Selection View Go Run Terminal Help Practical Assignments
import tkinter as tk
import random
import string

def generate_pin():
    return ''.join(random.choices(string.digits, k=10))

def submit_pin():
    pin = entry.get()
    if pin == generate_pin():
        submit_button.config(text="Success")
    else:
        submit_button.config(text="Invalid")

root = tk.Tk()
root.title("Online Registration System")
root.geometry("400x300")

frame = tk.Frame(root)
frame.pack(expand=True, fill="both")

for row in buttons:
    frame = tk.Frame(root)
    frame.pack(expand=True, fill="both")
    for btn_text in row:
        btn = tk.Button(frame, text=btn_text, font="Arial 18", relief="ridge", borderwidth=1)
        btn.pack(side="left", expand=True, fill="both")
        btn.bind("<Button-1>", click)

root.mainloop()
```

24. Draw a flowchart representing the logic of a basic online registration system.

