Lab Exercise

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

- 1.C program

```
//Hello World in c
  #include<stdio.h>
  Int main()
  {
          Printf("Hello World");
          Return 0;
  }
```

2.Python program

```
//Hello World in python

Print("Hello World")
```

Feature	**C Language**	**Python Language**
Compilation	Compiled language (needs a compiler)	Interpreted language (runs line-by-line)
Syntax Complexity	Requires semicolons, braces `{}`	Simple syntax, no semicolons or braces
Main Function	Requires a `main()` function to start execution	No need for a main function
Print Statement	Uses `printf()`	Uses `print()`
Header Files	Must include headers like `<stdio.h>`	No header files needed for basic output
Return Statement	Requires `return 0;` at the end of `main()`	Not required unless inside a defined function

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

➢

```
         Client (Browser / App)
              |
              |    Request: "GET /data"
              ↓
         [ TCP/IP Stack]
              ↓
         [ Router / Internet]
              ↓
         [ TCP/IP Stack]
              ↓
         Server (e.g., Web Server)
              |
              | Response: "Here is your data"
              ↑
         [ TCP/IP Stack]
              ↑
         [ Router / Internet]
              ↑
         [ TCP/IP Stack]
              ↑
         Client receives data
```

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

- ➢ Server code:

```
  # server.py
from http.server import SimpleHTTPRequestHandler, HTTPServer
```

```
# Create server on localhost and port 8000
server_address = ("localhost", 8000)
httpd = HTTPServer(server_address, SimpleHTTPRequestHandler)

print("Server running at http://localhost:8000")
httpd.serve_forever()
```

client code:

```
# client.py
import requests

response = requests.get("http://localhost:8000")
print("Server Response:")
print(response.text)
```

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

- 1.Digital Subscriber Line (DSL)

  Pros:

  - Widely available
  - Allows internet and phone use at the same time
  - Affordable for basic users

  Cons:

  - Speed depends on distance from service provider
  - Slower compared to modern options like fiber

2. Cable Internet

Pros:

- Faster than DSL
- Suitable for streaming and gaming
- Uses existing TV cable lines

Cons:

- Shared bandwidth can cause speed drops during peak hours
- Limited availability in rural areas

3. Fiber Optic

Pros:

- Very high speed (up to 1 Gbps or more)
- Low latency and highly reliable
- Great for heavy users (streaming, gaming, work-from-home)

Cons:

- Limited availability in some regions
- Installation may be expensive

4. Satellite Internet

Pros:

- Available in remote and rural areas
- Doesn’t require cable or phone lines

Cons:

- High latency (delay), not good for gaming or video calls
- Weather can affect signal quality
- Data caps and slower speeds

5. Wireless Internet (Mobile Data / Wi-Fi)

Pros:

- Convenient and portable
- Easy to set up
- Useful for smartphones and hotspots

Cons:

- Speed and reliability depend on signal strength
- May have data limits or be costly

6. Broadband over Power Lines (BPL)

Pros:

- Uses existing electrical infrastructure
- Easy access where other services are unavailable

Cons:

- Not widely available
- Interference issues can occur

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

➢ 1.Simulating an HTTP Request Using curl

Command:

```
curl http://example.com
```

Explanation:

- This command sends an HTTP GET request to the server at example.com.
- The server responds with the HTML content of the page.
- Useful for testing websites or APIs.

2. Simulating an FTP Request Using curl

Command (to download a file):

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

Explanation:

- Connects to an FTP server.
- Logs in with provided username and password.
- Downloads the file file.txt from the FTP server.

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

➢ 1.SQL Injection

- Problem: Hacker tricks the app to get into the database.
- Fix: Check and clean user input.

2. XSS (Cross-Site Scripting)

- Problem: Hacker puts bad code in a website that runs on other people's screens.
- Fix: Don't show user input directly. Clean it first.

3. Weak Login System

- Problem: Easy passwords or no security checks.
- Fix: Use strong passwords and add OTP or 2-step login.

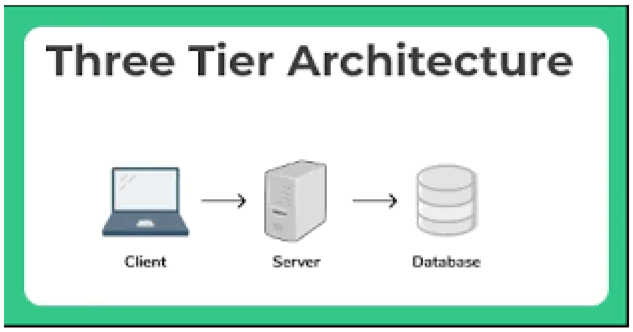
7) Identify and classify 5 applications you use daily as either system software Or application software.

➢

- Google Chrome – Application Software
- Microsoft Word – Application Software
- Windows 10 – System Software
- VLC Media Player – Application Software
- Antivirus (like Quick Heal) – System Software

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

➢



1.Presentation Layer (Client Tier)

- Purpose: User Interface
- Examples: Web browser, mobile app
- Technologies: HTML, CSS, JavaScript

2.Application Layer (Logic Tier)

- Purpose: Business Logic & Processing
- Examples: Server-side logic, API
- Technologies: Node.js, Python (Flask/Django), Java (Spring), PHP (Laravel)

3.Data Layer (Database Tier)

- Purpose: Data Storage and Management
- Examples: Relational or NoSQL Databases
- Technologies: MySQL, PostgreSQL, MongoDB

```
[ User / Browser ]

        |

  --------------------------

| 1. Presentation Layer  |   → HTML, CSS, JS

  --------------------------

        |
 --------------------------
| 2. Application Layer    |  → Backend logic (PHP,Python)
 --------------------------
            |
 --------------------------
| 3. Data Layer           |  → Database (MySQL, etc.)
 --------------------------
```

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

➢

1. Presentation Layer (Frontend / UI)

Role: This is what the user interacts with.

- User browses restaurants and food items
- Adds food to cart
- Enters delivery details
- Makes payment

Technologies Used:
HTML, CSS, JavaScript, React, Flutter (for mobile)

2. Business Logic Layer (Application Layer)

Role: This handles all decision-making and rules.

- Processes order and verify payment
- Applies discounts and taxes
- Matches user with nearby delivery agents
- Calculates estimated delivery time

Technologies Used:
Node.js, Java, PHP, Python

3. Data Access Layer (Database Layer)

Role: Deals with storing and retrieving data.

- Saves user profiles, orders, and payment info
- Fetches list of restaurants and menus
- Tracks real-time delivery status
- Stores feedback and reviews

Technologies Used:
MySQL, MongoDB, PostgreSQL

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

➢ Types of Software Environments:

1. Development Environment
   - Used by developers to write and build code
   - Contains IDEs, compilers, and debugging tools
   - Example: VS Code, Python, XAMPP
2. Testing Environment
   - Used by QA (testers) to test features
   - Isolated from development and production
   - Includes tools for automated/manual testing
   - Example: Selenium, Postman, JUnit
3. Production Environment
   - The live environment where real users access the application
   - Must be stable, secure, and monitored
   - Example: Hosted web server (Apache, Nginx), Cloud (AWS, Azure)

➢ Basic Virtual Machine Setup (Example using VirtualBox):

1. Install VirtualBox or VMware
2. Create a new virtual machine
    - Choose OS (e.g., Ubuntu or Windows)
    - Allocate RAM and disk space
3. Install a development stack
    - Example for web development:
        - Install Apache, MySQL, PHP (or use XAMPP)
        - Install code editor (e.g., VS Code)
4. Test a basic web page or script
    - Create a hello.php file
    - Run it in the browser from localhost

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

- 1. Write a Simple Code File

Create a simple file named hello.py:

```
# hello.py
print("Hello, GitHub!")
```

2. Create a Repository on GitHub

- Go to https://github.com
- Click New Repository
- Name it (e.g., first-code)
- Add a description (optional)
- Choose Public
- Click Create repository

3. Upload the Code Using Git (Command Line)

Open terminal or Git Bash:

```
git init
git add hello.py
git commit -m "Add hello.py"
git branch -M main
git remote add origin https://github.com/your-username/first-code.git
git push -u origin main
```

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

- Step 1: Create a GitHub Repository
  1. Go to https://github.com
  2. Click on "New" to create a new repository
  3. Enter a repository name (e.g., my-first-repo)
  4. (Optional) Add a description
  5. Choose Public or Private
  6. Click Create repository

Step 2: Prepare Your Project Locally

Create a folder and add a file (e.g., main.py):

python

Copy code

```
# main.py
print("This is my first commit!")
```

Step 3: Use Git to Commit and Push Code

Open Git Bash or Terminal, then run:

bash

Copy code

```
git init                    # Initialize Git in the folder
git add .                   # Stage all files
git commit -m "Initial commit"  # Commit changes with a message
git branch -M main          # Rename default branch to main
git remote add origin https://github.com/your-username/my-first-repo.git
git push -u origin main     # Push changes to GitHub
```

Replace your-username with your actual GitHub username.

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

- Objective

  To understand version control using GitHub and practice real-time collaboration on a basic project.

Tasks to Perform

1. Create a GitHub account by visiting https://github.com.
2. Set up your profile with your real name and profile photo.
3. Create a new repository named collab-project.
4. Add a README.md file describing the project.
5. Invite your classmate as a collaborator via repository settings.
6. Both team members should commit at least one file each.
7. Explore features like:
   - Issues
   - Pull requests
   - Commit history

Tools Required:

- GitHub account

- Web browser
- Basic internet connection

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

➢

Software Name	Category	Type	Purpose / Use
Windows 10 / 11	System Software	Operating System	Manages computer hardware & software
Ubuntu Linux	System Software	Operating System	Linux-based desktop/server OS
Android OS	System Software	Operating System	Mobile phone operating system
Google Chrome	Application Software	Web Browser	Browsing internet
Microsoft Word	Application Software	Word Processor	Creating and editing documents
VLC Media Player	Application Software	Media Player	Playing video and audio files
WhatsApp	Application Software	Communication Tool	Messaging, voice & video calling
Zoom	Application Software	Video Conferencing	Online meetings and webinars
Adobe Photoshop	Application Software	Graphic Editor	Photo editing and design
Microsoft Excel	Application Software	Spreadsheet Tool	Data entry, calculations, charts
WinRAR	Utility Software	File Compression Tool	Compressing and extracting files
CCleaner	Utility Software	System Cleaner	Cleaning junk files, optimizing PC
Avast Antivirus	Utility Software	Security Tool	Virus detection and system protection
Disk Defragmenter	Utility Software	Disk Optimization Tool	Improves hard drive performance
File Explorer	Utility Software	File Manager	Browsing and managing files/folders

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

- 1.Cloning a Repository
    - Use git clone to download a remote repository to your local machine.
    - Example:

bash

code

git clone https://github.com/username/repository-name.git

2.Creating a Branch

- Create a new branch to add features without affecting the main code.
- Example:

bash

code

git checkout -b feature-branch

3.Making Changes

- Edit files, commit the changes using git commit, and push to the new branch.

4.Merging Branches

- Switch to the main branch and merge the feature branch into it.
- Example:

bash

code--

git checkout main

git merge feature-branch

5.Resolve Merge Conflicts (if any)

- Practice conflict resolution when Git highlights file conflicts.

⚙ Tools Required

- Git installed on your computer
- GitHub account with a repository
- Command-line interface or Git GUI (like Git Bash, GitHub Desktop)

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

- ➢ Application software refers to programs designed to help users perform specific tasks efficiently. From writing documents to managing finances or editing photos, application software plays a crucial role in increasing personal and professional productivity.

□ Types of Application Software

Below are common types of application software and their usage:

1. Word Processing Software

- Examples: Microsoft Word, Google Docs, LibreOffice Writer
- Use: Creating, editing, and formatting text documents.
- Productivity Impact: Speeds up writing, editing, spell-checking, and document sharing.

2. Spreadsheet Software

- Examples: Microsoft Excel, Google Sheets, LibreOffice Calc
- Use: Data analysis, financial calculations, and data visualization.
- Productivity Impact: Automates complex calculations, budgeting, and tracking through formulas and charts.

3. Presentation Software

- Examples: Microsoft PowerPoint, Google Slides, Keynote
- Use: Creating visual presentations for meetings or lectures.
- Productivity Impact: Makes communication of ideas clearer and more engaging.

4. Database Management Software

- Examples: Microsoft Access, MySQL, Oracle
- Use: Storing and managing large volumes of structured data.
- Productivity Impact: Enables quick search, update, and organization of data.

5. Graphics and Design Software

- Examples: Adobe Photoshop, CorelDRAW, Canva
- Use: Editing images, designing logos, posters, and visual content.
- Productivity Impact: Enhances creativity and reduces time needed to produce visual materials.

6. Communication Software

- Examples: Zoom, Microsoft Teams, WhatsApp, Gmail
- Use: Video conferencing, email, and messaging.
- Productivity Impact: Improves collaboration, remote work, and real-time communication.

7. Web Browsers

- Examples: Google Chrome, Mozilla Firefox, Microsoft Edge
- Use: Accessing information online.
- Productivity Impact: Enables quick access to online tools, research, and cloud applications.

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

➢

$$\begin{tikzcd}
\text{1. Planning} \arrow[d] \\
\text{2. Requirements Gathering \& Analysis} \arrow[d] \\
\text{3. Design (Architecture, UI, DB)} \arrow[d] \\
\text{4. Implementation (Coding \& Development)} \arrow[d] \\
\text{5. Testing (Unit, System, QA)} \arrow[d] \\
{}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{6. Deployment} \arrow[d] \\
\text{7. Maintenance}
\end{tikzcd}$$

(Bug fixing, updates)

Explanation of Phases:

1. **Planning:** Define the scope, goals, and timeline.
2. **Requirements Analysis:** Understand and document what the system must do.
3. **Design:** Create system architecture, UI, and database design.
4. **Implementation:** Developers write the code.
5. **Testing:** Find and fix bugs before going live.
6. **Deployment:** Release the software to users.
7. **Maintenance:** Update, support, and improve the software.

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

➢ Tasks to Perform

1. Define the purpose and scope of the system.
2. Identify the functional requirements (what the system should do).
3. Identify the non-functional requirements (system qualities like performance, security).
4. Present the specification in standard SRS format.

📝 Sample Requirement Specification Document

◈ 1. Introduction

- Purpose:
  To manage books, members, and borrowing activities in a digital format.
- Scope:
  The system will allow librarians to add/remove books, register members, issue/return books, and generate reports.

◈ 2. Functional Requirements

- The system shall allow the librarian to:
  - Add, delete, and update book records.

    - Register and manage members.
    - Issue books to members.
    - Return books from members.
    - Generate overdue fine reports.
- The system shall display:
    - Available and borrowed books.
    - Member transaction history.
    - Due date alerts.

🔷 3. Non-Functional Requirements

- Usability: User-friendly UI for easy navigation.
- Reliability: System should handle simultaneous users and maintain data consistency.
- Security: Login credentials required for librarian and staff access.
- Performance: The system should perform all operations within 2 seconds.
- Portability: Should work on web browsers and desktop platforms.

🔷 4. Assumptions

- Users have basic computer literacy.
- Database is regularly backed up.

⚙ Tools Required

- Word processor (e.g., MS Word, Google Docs)
- Internet (optional, for reference templates)

📝 Learning Outcome

After completing this lab, students will:

- Understand how to define functional and non-functional requirements
- Gain practice in writing technical documents
- Learn how proper specification prevents software development errors

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

- ➢ Tasks to Perform

1. Identify key user roles and system actors (e.g., Customer, Admin).
2. List core functional requirements and explain their purpose.
3. Draw a functional block diagram (optional) for better understanding.

📋 Functional Requirements of Online Shopping System

🔹 1. User Registration & Login

- Users must be able to register and securely log in.
- Forgot password and user authentication features included.

🔹 2. Product Browsing and Search

- Users can browse by category, search for products using keywords, and filter results.

🔹 3. Shopping Cart

- Users can add/remove products, view totals, and update quantities.

🔹 4. Checkout and Payment

- System calculates total price with taxes and shipping.
- Supports payment gateways like UPI, Credit/Debit Cards, Net Banking.

🔹 5. Order Management

- Users can view order history, current status (shipped, delivered), and cancel orders.

🔹 6. Admin Functionalities

- Add/update/delete product listings
- Manage inventory, users, and process orders

🔹 7. Feedback and Reviews

- Customers can leave product ratings and reviews.

□ Optional Functional Block Diagram

A diagram showing the flow between:
User → Product Search → Cart → Checkout → Payment → Order Confirmation

⚙ Tools Required

- Word processor
- Diagramming tool (for functional block diagram, optional)

📝 Learning Outcome

After completing this lab, students will be able to:

- Identify key functionalities of real-world systems
- Perform structured analysis of a complex application
- Understand how to break down large systems into manageable features

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

➢ Tasks to Perform

1. Identify the main system components and user roles.
2. Design a basic architecture diagram.
3. Describe the role of each component and how data flows through the system.

□ Architecture Components

🔹 1. Frontend (User Interface)

- Customer App: Browse restaurants, place orders, track delivery.
- Restaurant Panel: Accept/prepare orders, update status.
- Delivery App: Accept delivery tasks, update real-time location.

🔹 2. Backend (Application Server)

- Handles:
  - Order placement logic
  - Authentication and user data
  - Payment integration
  - Notification system (push/SMS/email)
  - Order status updates

🔹 3. Database Layer

Stores:

- User data (login, address, orders)
- Restaurant menus and availability
- Payment history and reviews
- Delivery logs

🔹 4. Payment Gateway API

- Securely processes transactions via UPI, cards, wallets, etc.

🔹 5. Real-Time Tracking System

- Uses GPS and mapping APIs (e.g., Google Maps)
- Tracks delivery location
- Shows ETA to customers

🔹 6. Notification System

- Sends order confirmations, delivery status, offers, etc.

📊 Sample Architecture Diagram (Text Representation)

css

Copy code

```
[Customer App] ——> [Backend Server] <—— [Restaurant Panel]
       |                  |
       ▼                  ▼
 [Payment Gateway]    [Database Layer]
       |                  ▲
       ▼                  |
[Real-time GPS] <—— [Delivery App]
```

⚙ Tools Required

- Drawing tool (Draw.io / Lucidchart / MS PowerPoint)

- Word processor for documentation

📝 Learning Outcome

After completing this lab, students will:

- Understand the structure of multi-user, real-time systems
- Be able to create and explain a basic system architecture
- Recognize the importance of APIs, data storage, and user interfaces in modern apps

21)Develop test cases for a simple calculator program

➢ Tasks to Perform

1. Identify the calculator functions to be tested.
2. Define input values, expected output, and conditions.
3. Organize test cases into a test case table.

□ Calculator Functionalities to Test

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Handling of invalid inputs
- Division by zero

⚙ Tools Required

- Calculator Program (Python/C/Java/Any Language)
- Word processor or spreadsheet software to document test cases

📝 Learning Outcome

After completing this lab, students will:

- Understand the importance of test cases in software quality assurance
- Be able to write effective test cases for simple programs
- Learn how to validate correct and incorrect input handling

22)Document a real-world case where a software application required critical maintenance

- ➢ Tasks to Perform

1. Research a known software maintenance case.
2. Describe the problem, its cause, and the maintenance performed.
3. Summarize the outcome and lessons learned.

📄 Case Study: WhatsApp Outage – October 2022

🔹 1. Background

WhatsApp, the popular messaging application owned by Meta, faced a global outage on 25th October 2022. Users were unable to send or receive messages for over two hours.

🔹 2. Problem Description

- Messages were stuck on the "clock" icon.
- Groups and private chats were unresponsive.
- Web version also failed to connect.
- The issue impacted millions of users worldwide.

🔹 3. Cause

- Internal server configuration changes triggered a major communication breakdown between WhatsApp servers.
- Load balancing failed due to improper update deployment.

🔹 4. Maintenance Actions Taken

- The engineering team rolled back the latest deployment.
- Reconfigured server communication modules.
- Conducted an emergency round of system health checks and network traffic balancing.

🔹 5. Outcome

- Services were gradually restored within 2.5 hours.
- Meta issued a public apology and promised enhanced monitoring.
- Internal deployment processes were revised to include stricter testing phases.

⚙ Tools Required

- Internet connection for research
- Word processor for report writing

📝 Learning Outcome

After completing this lab, students will:

- Gain awareness of real-world maintenance challenges
- Understand how maintenance impacts users and business reputation

Learn best practices in error recovery and rollback strategy

23)Create a DFD for a hospital management system

Tasks to Perform

1. Identify key processes and external entities in the hospital system.
2. Create a Level 0 DFD (Context Diagram).
3. Expand into a Level 1 DFD showing detailed interactions.

📊 Level 0 DFD (Context Diagram)

External Entities:

- Patient
- Doctor
- Receptionist
- Admin

Processes:

- Hospital Management System

Data Flows:

- Patient provides registration details
- Doctor provides diagnosis
- Receptionist schedules appointments

- Admin manages records

Code---

[Patient] → (HMS) ← [Doctor]

[Receptionist] → (Hospital Management System) ← [Admin]

💹 Level 1 DFD (Detailed Process Breakdown)

Processes:

1. Patient Registration
2. Appointment Scheduling
3. Medical Diagnosis
4. Billing and Discharge
5. Report Generation

Data Stores:

- Patient Records
- Appointment Database
- Billing Info
- Medical History

Example Flow:

scss

code--

[Patient] → (1. Patient Registration) → [Patient Records]

[Receptionist] → (2. Appointment Scheduling) → [Appointment DB]

[Doctor] → (3. Medical Diagnosis) ↔ [Medical History]

(HMS) → (4. Billing & Discharge) → [Billing Info]

□ Tools Required

- Diagram tool (Draw.io / Lucidchart / Paper sketch)
- Word processor for documentation

📝 Learning Outcome

After completing this lab, students will:

- Understand the structure of DFDs and how to read/create them
- Learn to break down a real-world system into logical data processes
- Be able to model data flow for complex systems like healthcare software

24)Build a simple desktop calculator application using a GUI library

Tasks to Perform

1. Design a calculator GUI with buttons for digits (0-9), operations (+, –, ×, ÷), clear, and equals.
2. Implement logic to handle button clicks and perform operations.
3. Display results and handle invalid inputs (e.g., division by zero).

□ Suggested Tech Stack

- Language: Python (Recommended)
- GUI Library: Tkinter

📐 Design Notes

- Use frames to organize buttons into rows
- Validate inputs and handle edge cases
- UI should be responsive and user-friendly

⚙ Tools Required

- Python 3.x
- Tkinter (comes built-in with Python)
- Code editor (VS Code / PyCharm / IDLE)

📝 Learning Outcome

After completing this lab, students will:

- Understand GUI event handling and layout design
- Be able to create interactive desktop apps

- Learn how to integrate logic with GUI controls

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

- ➢ Tasks to Perform

1. Identify the sequence of steps a user follows in an online registration form.
2. Define decision points such as validation and duplication check.
3. Draw a flowchart using standard flowchart symbols.

□ Flowchart Logic Description

1. Start
2. Display Registration Form
3. User Inputs Details
4. Validate Required Fields
    - If Invalid → Show Error → Go to Step 3
    - If Valid → Proceed
5. Check If User Already Exists
    - If Yes → Show "User Exists" Message → End
    - If No → Proceed
6. Store User Data in Database
7. Show Registration Success Message
8. End

🔁 Flowchart (Text Representation)

code

```
[Start]
  ↓
[Display Registration Form]
  ↓
[User Enters Details]
```

```
  ↓
[Validate Inputs]
  ↓
 ┌──────────────────┐
 │ Inputs Valid? │
 └───────┬──────────┘
     │Yes       No
     ↓        ↙
[Check If User Exists] ← [Show Error]
     ↓
 ┌──────────────────┐
 │ User Exists?  │
 └───────┬──────────┘
     │Yes       No
     ↓        ↙
[Show Exists Msg] ← [Store in Database]
     ↓
  [End] ← [Show Success]
```

□ Tools Required

- Paper & Pen (for manual diagram)
- OR
- Diagram Tools (Draw.io, Lucidchart, Creately, etc.)

📝 Learning Outcome

After completing this lab, students will:

- Understand how to visualize decision-making in a system
- Learn flowchart components like decision, process, and input/output
- Gain experience mapping real-world processes into diagrams