Lab Exercise

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

1. Python Program:

```
# Hello World in Python
Print ("Hello, World!")
```

2. C Program:

```
        // Hello World in C
       #include <stdio.h>
      int main() {
     printf("Hello, World!");
   return 0;
}
```

Feature	***Python***	***C***
Syntax Simplicity	*Very simple*	*More structured*
Output Statement	*print("Hello, World!")*	*printf("Hello, World!");*
Main Function Required	*No*	*Yes*
Semicolon Needed	*No*	*Yes*
Header Files	*Not required*	*Required (include stdio.h)*

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]

↑

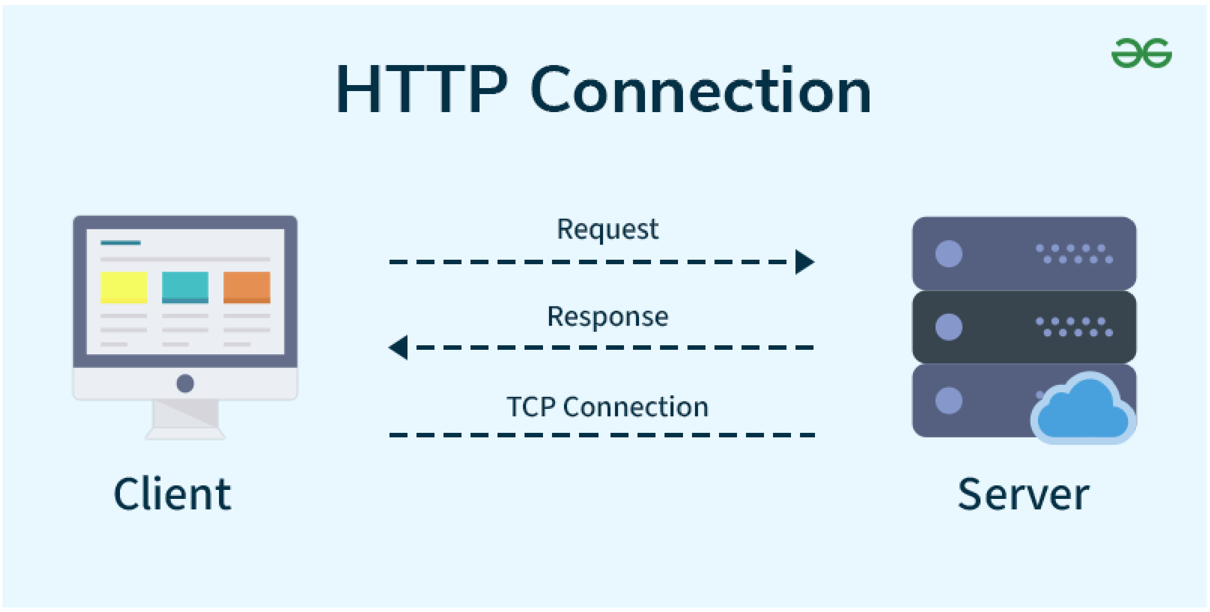
[TCP/IP Stack]

↑

Client receives data

Client sends request → internet → server → server sends response → internet → client

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



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.

```
+------------------------------+
|   Presentation Layer        |
  (Browser / Mobile App)
|   HTML, CSS, JavaScript     |
+--------------|---------------+
               |
               v
+------------------------------+
|   Application Layer         |
|   (Web Server / API)     |
|   Business Logic          |
|   e.g., Node.js, Django     |
+--------------|---------------+
               |
               v
+------------------------------+
|   Data Layer              |
|   (Database Server)       |
|   e.g., MySQL, MongoDB      |
+------------------------------+
```

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.

1) Development Environment

- *Used by developers to write and debug code.*
- *Frequently updated with new features or bug fixes.*
- *Often runs locally on a developer's machine or on a dedicated dev server.*
- *Not stable; can have incomplete features.*
- *Tools: IDE (e.g., VS Code), local databases, mock services.*

Example:
A developer writes code on their laptop, tests it locally, then pushes to a shared development server.

2) Testing Environment

- *Used by QA (Quality Assurance) teams.*
- *Has code that is complete and ready for validation.*
- *Simulates production as closely as possible to catch bugs before deployment.*
- *Contains real-like data (but not actual user data).*

- *Automated tests and manual tests run here.*

Example:
QA team tests if the checkout process works, if discounts apply correctly, and if the system handles invalid inputs.

3) Production Environment

- *Live environment used by real end users.*
- *Must be highly stable, secure, and monitored.*
- *Only thoroughly tested code is deployed here.*
- *Connected to live databases with real user data.*

Example:
Customers shop on the live e-commerce website; data entered here affects actual orders and stock.

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.
```

Summary:

- *You created a GitHub repository*
- *Committed code using Git*
- *Pushed it to GitHub successfully*

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.

Objective

To recognize and categorize commonly used software based on their function and purpose within a computing environment.

Tasks to Perform

1. *Identify 9–12 software applications you use regularly on your computer or smartphone.*
2. *Organize the software into one of the three types:*
    - *System Software*
    - *Application Software*
    - *Utility Software*
3. *Present the information in a table format with proper headings.*

Table Format

Software Name	*Category*	*Description / Purpose*
Windows 11	*System Software*	*Manages hardware and provides system interface*
Android OS	*System Software*	*Mobile operating system*
MS Word	*Application Software*	*Word processing and document creation*
Google Chrome	*Application Software*	*Internet browsing*
VLC Media Player	*Application Software*	*Playing audio and video files*
Tally ERP	*Application Software*	*Accounting and financial management*
WinRAR	*Utility Software*	*Compressing and extracting files*
CCleaner	*Utility Software*	*Cleaning junk files and optimizing performance*
Antivirus (e.g., Avast)	*Utility Software*	*Protecting against malware and viruses*

Tools Required

- *Access to your device's installed software list*
- *Pen-paper or text editor for writing*
- *Internet (optional, for researching unfamiliar software)*

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

Objective

To understand and apply the basic operations of Git for version control, including cloning a repository, creating branches, and merging code.

Tasks to Perform

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

1. Introduction

Application software is a type of computer program designed to help users perform specific tasks such as word processing, web browsing, or graphic design. Unlike system software, which runs the computer itself, application software enables users to accomplish practical goals.

2. Types of Application Software

1. Word Processing Software

- ***Examples:*** *Microsoft Word, Google Docs, LibreOffice Writer*
- ***Use:*** *Creating and editing documents, reports, letters.*
- ***Productivity Benefit:*** *Speeds up writing, editing, and formatting tasks. Built-in spellcheck and templates save time.*

2. Spreadsheet Software

- ***Examples:*** *Microsoft Excel, Google Sheets*
- ***Use:*** *Calculations, data analysis, budgeting, charts.*
- ***Productivity Benefit:*** *Automates math functions, organizes large datasets, supports decision-making with graphs and formulas.*

3. Presentation Software

- ***Examples:*** *Microsoft PowerPoint, Google Slides, Canva*
- ***Use:*** *Creating slideshows for meetings, lessons, and reports.*
- ***Productivity Benefit:*** *Enhances communication and makes ideas clearer using visuals and animations.*

4. Database Management Software (DBMS)

- ***Examples:*** *Microsoft Access, MySQL, Oracle DB*
- ***Use:*** *Storing, retrieving, and managing data efficiently.*
- ***Productivity Benefit:*** *Centralized data handling reduces errors, improves organization, and allows data sharing.*

5. Graphic Design Software

- ***Examples:*** *Adobe Photoshop, CorelDRAW, Canva*
- ***Use:*** *Designing images, logos, brochures, social media posts.*
- ***Productivity Benefit:*** *Boosts creativity, allows fast revisions, and produces professional-looking visuals.*

6. Web Browsers

- ***Examples:*** *Google Chrome, Mozilla Firefox, Microsoft Edge*
- ***Use:*** *Accessing websites, web apps, research, and online communication.*
- ***Productivity Benefit:*** *Provides fast access to information, tools, and cloud-based apps like Google Drive or Trello.*

7. Email and Communication Tools

- ***Examples:*** *Microsoft Outlook, Gmail, Slack, Zoom*
- ***Use:*** *Sending emails, chatting, video conferencing, and task coordination.*

- ***Productivity Benefit:*** *Improves team collaboration, supports remote work, and reduces delays in communication.*

8. Project Management Software

- ***Examples:*** *Trello, Asana, Microsoft Project*
- ***Use:*** *Tracking tasks, timelines, team assignments.*
- ***Productivity Benefit:*** *Helps teams meet deadlines, monitor progress, and prioritize work effectively.*

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

Objective

To visually represent the phases of the Software Development Life Cycle (SDLC) using a standard flowchart and understand the role of each phase in structured software development.

Tasks to Perform

1. *Study the six basic phases of SDLC.*
2. *Use a diagram tool (e.g., Draw.io, Lucidchart, MS Visio, or even pen-paper) to design a flowchart.*
3. *Ensure each SDLC phase is shown with correct flow and relationships.*

Phases of SDLC to Include

- *1. Requirement Analysis*
- *2. System Design*
- *3. Implementation (Coding)*
- *4. Testing*
- *5. Deployment*
- *6. Maintenance*

Sample Flowchart Structure

csharp

Copy code

```
[Start]
   ↓
[Requirement Analysis]
   ↓
[System Design]
   ↓
[Implementation]
   ↓
[Testing]
   ↓
[Deployment]
   ↓
[Maintenance]
   ↓
[End]
```

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

1. Purpose

To manage library operations like book issue/return, member registration, and book catalog maintenance.

2. Scope

- *Book management*
- *Member management*
- *Issue/Return system*
- *Fine calculation*

3. Users

- ***Admin**: Manages books and users*
- ***Member**: Borrows and returns books*

4. Functional Requirements

- *Admin can:*
  - *Add/edit/delete books and members*
  - *Issue and return books*
  - *Track fines and view reports*
- *Member can:*

- *Register/login*
- *Search books*
- *Request issue/return*
- *View borrow history*

5. Non-Functional Requirements

- *Easy-to-use interface*
- *Secure login system*
- *Reliable and fast performance*
- *Data backup*

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.*

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

1. Introduction

A food delivery app allows users to order food from restaurants and have it delivered to their location. The system must manage users, restaurants, menus, orders, delivery tracking, and payments.

2. Basic System Architecture (Layered Structure)

```
┌─────────────────────────┐
│  1. Presentation Layer │
└─────────────────────────┘
 │ Mobile App (User & Delivery)
│ Web App (Admin, Restaurant)
             ▼
┌─────────────────────────┐
│   2. Application Layer  │
└─────────────────────────┘
       │ Business Logic
     │ Order Management
       │ Authentication
     │ Payment Processing
             ▼
┌─────────────────────────┐
│     3. Data Layer       │
└─────────────────────────┘
         │ User Data
       │ Restaurant Data
      │ Orders & Payments
```

3. Components of the Architecture

1. Frontend (Presentation Layer)

- ***User App**: Browse restaurants, place orders, track deliveries*
- ***Delivery App**: Receive delivery requests, update order status*
- ***Restaurant Panel**: Manage menu, orders, availability*
- ***Admin Dashboard**: Monitor overall system*

2. Backend (Application Layer)

- ***Authentication Module**: Login, registration, session control*
- ***Menu Management**: Add/update food items*
- ***Order Management**: Handles order creation, status updates*
- ***Payment Gateway**: Secure online transactions*
- ***Notification System**: Sends order updates (SMS/email/push)*

3. Database (Data Layer)

- ***Users**: Name, address, contact info*
- ***Restaurants**: Info, menus, ratings*
- ***Orders**: Items, price, status*
- ***Payments**: Transaction details*
- ***Delivery Staff**: Location, status*

21. Develop test cases for a simple calculator program

Addition Test Cases

1. *Add two positive numbers: 5 + 3 → **Expected Output:** 8*
2. *Add a positive and a negative number: -4 + 6 → **Expected Output:** 2*
3. *Add two negative numbers: -2 + (-3) → **Expected Output:** -5*
4. *Add two decimal numbers: 2.5 + 3.1 → **Expected Output:** 5.6*

Subtraction Test Cases

5. *Subtract smaller from larger: 9 - 4 → **Expected Output:** 5*
6. *Subtract larger from smaller: 4 - 9 → **Expected Output:** -5*
7. *Subtract decimal values: 5.5 - 2.0 → **Expected Output:** 3.5*

Multiplication Test Cases

8. *Multiply two positive numbers: 6 * 7 → **Expected Output:** 42*
9. *Multiply by zero: 7 * 0 → **Expected Output:** 0*
10. *Multiply two negative numbers: -3 * -2 → **Expected Output:** 6*

Division Test Cases

11. *Divide two positive numbers: 8 / 2 → **Expected Output:** 4*
12. *Divide with decimal result: 7 / 2 → **Expected Output:** 3.5*
13. *Divide by zero: 7 / 0 → **Expected Output: Error** or **Infinity***
14. *Divide negative by positive: -10 / 2 → **Expected Output:** -5*

Invalid Input Test Cases

15. *Add non-numeric input: "a" + 3 → **Expected Output: Error***
16. *Empty input: "" + 2 → **Expected Output: Error***
17. *Special characters: # + 4 → **Expected Output: Error***

These test cases cover:

- *Basic operations (Add, Subtract, Multiply, Divide)*
- *Edge cases (Zero, Negative numbers, Decimals)*
- *Error handling (Invalid or non-numeric input)*

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

1. Background:

*WhatsApp faced a global outage on **25th October 2022**, affecting millions of users who couldn't send or receive messages.*

2. Problem:

- *Messages were stuck and not delivered*
- *App and web version became unresponsive*
- *Affected private and group chats*

3. Cause:

*A faulty **server configuration update** caused internal communication and load balancing failures.*

4. Maintenance Performed:

- *Rolled back the update*
- *Reconfigured server modules*

- *Ran emergency system checks*

5. Outcome:

- *Service restored in **2.5 hours***
- *Meta apologized and improved internal deployment/testing processes*

23. Create a DFD for a hospital management system

Objective

To understand and visualize the flow of data within a Hospital Management System (HMS) by creating a Level 0 and Level 1 DFD that includes key entities, processes, and data stores.

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:

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*

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

Objective

To design and develop a desktop calculator with basic arithmetic functionality (Addition, Subtraction, Multiplication, Division) using a Graphical User Interface (GUI) library such as Tkinter (Python), JavaFX (Java), or WinForms (C#).

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.

Objective

To understand the logical flow of user interactions in an online registration system and visualize the process using a flowchart diagram.

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*