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

**A. C Language**

#include <stdio.h>

void main() {

printf("Hello, World!\n");

}

**B. C++ Language**

#include <iostream>

int main() {

std::cout << "Hello, World!" << std::endl;

return 0;

}

**Comparison:**

| **Feature** | **C** | **C++** |
| --- | --- | --- |
| Header File | <stdio.h> | <iostream> |
| Output Method | printf() | std::cout |
| Entry Point | void main() | int main() |
| Syntax Style | Procedural | Object-Oriented |
| Namespace | Not used | Uses std:: namespace |

**2. Diagram: Data Transmission from Client to Server**

Client (Browser)

|

| HTTP Request

v

+--------------------+

| Local Network |

+--------------------+

|

| Internet Backbone (Routers, Switches)

v

+--------------------+

| Web Server |

+--------------------+

|

| HTTP Response

v

Client (Browser displays response)

**Explanation:**

* **Client:** Initiates a request (e.g., typing a URL).
* **DNS Resolution:** Resolves domain name to IP.
* **TCP/IP Layer:** Breaks request into packets.
* **HTTP Layer:** Wraps the request in protocol headers.
* **Server:** Receives, processes, and sends response.

**3. Simple HTTP Client-Server Communication in Python**

**A. Server Code (Python)**

from http.server import BaseHTTPRequestHandler, HTTPServer

class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):

def do\_GET(self):

self.send\_response(200)

self.send\_header('Content-type', 'text/plain')

self.end\_headers()

self.wfile.write(b'Hello from the server!')

def run(server\_class=HTTPServer, handler\_class=SimpleHTTPRequestHandler):

server\_address = ('localhost', 8080)

httpd = server\_class(server\_address, handler\_class)

print('Starting server on http://localhost:8080')

httpd.serve\_forever()

if \_\_name\_\_ == '\_\_main\_\_':

run()

**B. Client Code (Python)**

import requests

def fetch\_data():

url = 'http://localhost:8080'

response = requests.get(url)

print('Status Code:', response.status\_code)

print('Response Text:', response.text)

if \_\_name\_\_ == '\_\_main\_\_':

fetch\_data()

**How to Run:**

1. Save and run server.py in one terminal: python server.py
2. Save and run client.py in another terminal: python client.py

**4. Comparison of Internet Connection Types**

| **Type** | **Pros** | **Cons** |
| --- | --- | --- |
| **DSL** | Available in rural areas, affordable, uses phone lines | Slower speeds, distance affects quality |
| **Cable** | Faster than DSL, reliable, supports gaming | Shared bandwidth = slower during peak hours |
| **Fiber** | Very high speed (up to 1 Gbps+), reliable, low latency | Expensive installation, limited to urban areas |
| **Satellite** | Available almost anywhere, good for remote areas | High latency, affected by weather, expensive |
| **Mobile (4G/5G)** | Portable, fast with 5G, easy setup | Data limits, depends on signal strength and location |

**5. Simulate HTTP and FTP Requests Using curl**

**A. HTTP Requests**

1. **GET Request**

curl http://example.com

1. **POST Request**

curl -X POST -d "username=test&password=1234" http://example.com/login

1. **GET with Headers**

curl -H "Accept: application/json" http://example.com/api/data

1. **Download File**

curl -O http://example.com/file.zip

1. **View Headers**

curl -I http://example.com

**B. FTP Requests**

1. **Download File (Anonymous)**

curl ftp://speedtest.tele2.net/1MB.zip -O

1. **Download File (Authenticated)**

curl -u username:password ftp://ftp.example.com/file.txt -O

1. **Upload File**

curl -T myfile.txt -u username:password ftp://ftp.example.com/uploads/

Here is a detailed elaboration for questions 6 to 10 with **code examples**, **explanations**, and **practical commands** where applicable:

**6. Identify and classify 5 applications as system or application software**

| **Software Name** | **Type** | **Explanation** |
| --- | --- | --- |
| Windows OS | System Software | Manages hardware resources, file system, process scheduling, device I/O. |
| Google Chrome | Application Software | Used to browse the internet. Relies on OS to work. |
| Microsoft Word | Application Software | Word processor for creating/editing documents. |
| VLC Media Player | Application Software | Plays video/audio files. Uses codecs and interface built atop OS. |
| Slack (or WhatsApp) | Application Software | Communication platform; includes chat, calls, and media sharing. |

**Code to simulate application detection in Python:**

import platform

import os

print("System Information:")

print("Operating System:", platform.system()) # System Software

# Simulate Application Software Usage

apps = ["Google Chrome", "Microsoft Word", "Slack", "VLC Player"]

print("\nApplication Software Installed:")

for app in apps:

print("-", app)

**7. Design a basic Three-Tier Architecture Diagram for Web Application**

**Diagram (Text Format)**

+-----------------------------+

| Presentation Layer |

| (React / Angular / HTML) |

+-----------------------------+

|

v

+-----------------------------+

| Application Layer |

| (Node.js / Django API) |

+-----------------------------+

|

v

+-----------------------------+

| Data Layer |

| (MySQL / MongoDB / Oracle) |

+-----------------------------+

**Code Components:**

**A. Presentation Layer (HTML)**

<!-- index.html -->

<h2>Search Book</h2>

<input type="text" placeholder="Enter book title">

<button onclick="searchBook()">Search</button>

**B. Application Layer (Node.js)**

// server.js

const express = require('express');

const app = express();

app.get('/books', (req, res) => {

res.send([{ title: 'The Alchemist', author: 'Paulo Coelho' }]);

});

app.listen(3000, () => console.log('Server running on port 3000'));

**C. Data Layer (MongoDB Example with Mongoose)**

const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/bookdb');

const Book = mongoose.model('Book', { title: String, author: String });

Book.find({}, (err, books) => {

console.log("Books from DB:", books);

});

**8. Case Study: Bookstore Software (Presentation, Logic, Data Layers)**

**1. Presentation Layer (Frontend)**

<!-- HTML UI -->

<div class="book-card">

<h3>The Alchemist</h3>

<p>By Paulo Coelho</p>

<button>Add to Cart</button>

</div>

**2. Business Logic Layer (Node.js Example)**

// Logic to add a book to cart

function addToCart(userId, bookId) {

if (bookInStock(bookId)) {

cartService.add(userId, bookId);

return "Book added!";

} else {

return "Out of stock";

}

}

**3. Data Access Layer (MySQL + SQL)**

-- SQL Example

SELECT \* FROM books WHERE title LIKE '%alchemist%';

// MongoDB Example with Mongoose

Book.find({ title: /alchemist/i }, (err, books) => {

console.log("Search result:", books);

});

**9. Software Environments + VM Setup**

**A. Types of Environments**

| **Environment** | **Purpose** | **Tools Used** |
| --- | --- | --- |
| Development | Build & test new code | VS Code, Node.js, Local DB |
| Testing | Test under production-like setup | Postman, Selenium, JMeter |
| Production | Live environment | Apache, Nginx, Docker, Monitoring |

**B. Set Up Development Environment in VM**

**1. Install Ubuntu on VirtualBox**

* Allocate: 2GB RAM, 10GB Storage
* Use Ubuntu ISO to install

**2. Install Node.js Stack**

sudo apt update

sudo apt install nodejs npm -y

node -v

npm -v

**3. Create Sample App**

mkdir myapp && cd myapp

npm init -y

npm install express

**app.js:**

const express = require('express');

const app = express();

app.get('/', (req, res) => res.send('Hello from the VM!'));

app.listen(3000, () => console.log('Server running on port 3000'));

**Run:**

node app.js

Visit http://localhost:3000 (or use port forwarding in VM settings).

**10. Write and Upload First Source Code to GitHub**

**Step-by-Step**

**A. Create hello.py**

# hello.py

print("Hello, GitHub!")

**B. Create New Repository**

1. Go to [GitHub](https://github.com/)
2. Click on ➕ → New Repository
3. Name: first-code-upload

**C. Push Using Git**

# Clone the new empty repo

git clone https://github.com/your-username/first-code-upload.git

cd first-code-upload

# Copy and commit your code

cp /path/to/hello.py .

git add hello.py

git commit -m "Initial commit: Added hello.py"

git push origin main

**D. Configure Git (Optional)**

git config --global user.name "Your Name"

git config --global user.email "you@example.com"

**E. Verify**

* Visit the repo to confirm the uploaded hello.py file.

Here is an **elaborated and expanded version of answers for questions 11 to 15**, with explanations, examples, and real-life applications:

**11. Create a GitHub Repository and Document How to Commit and Push Code Changes**

**✅ Prerequisites:**

* GitHub account
* Git installed locally: https://git-scm.com/
* Text/code editor (e.g., VS Code, Sublime)

**🔧 Step 1: Create a New Repository on GitHub**

1. Go to [GitHub](https://github.com/).
2. Click on **‘+’ ➜ New Repository**.
3. Fill in details:
   * **Repository Name**: my-first-repo
   * **Description** (Optional): "My first GitHub project"
   * **Visibility**: Public or Private
4. (Optional) Uncheck “Initialize this repository with a README”
5. Click **Create Repository**.

**💻 Step 2: Set Up the Repository Locally**

**Using Terminal:**

# Clone the repository

git clone https://github.com/your-username/my-first-repo.git

# Navigate into it

cd my-first-repo

# Create a Python file

echo "print('Hello GitHub!')" > hello.py

**📝 Step 3: Commit and Push Code**

# Check modified files

git status

# Add file to staging

git add hello.py

# Commit the file with a message

git commit -m "Add hello.py with hello message"

# Push code to the main branch

git push origin main

**🌐 Step 4: View on GitHub**

* Go to your repository page on GitHub.
* You’ll now see the file hello.py with your commit message.

**12. Create a Student Account on GitHub and Collaborate on a Project**

**🎓 Step 1: Apply for GitHub Student Developer Pack**

1. Visit: [GitHub Student Pack](https://education.github.com/pack)
2. Sign in with GitHub or create an account.
3. Submit:
   * College email (e.g., [you@university.edu](mailto:you@university.edu))
   * Proof: Student ID, Admission letter
   * School name and graduation year
4. Approval time: 1–5 days

**🤝 Step 2: Collaborate with a Classmate**

**A. Create a Collaborative Repository**

1. Go to GitHub ➜ “+” ➜ New Repository
2. Name: student-collab-project
3. Initialize with:
   * README file
   * .gitignore (Node, Python, etc.)
4. Click **Create Repository**

**B. Add Collaborator**

1. In repo settings ➜ **Manage Access**
2. Click **Invite Collaborator**
3. Enter your classmate’s GitHub username
4. They’ll receive an invitation via email.

**🧑‍💻 Step 3: Clone and Start Coding**

git clone https://github.com/your-username/student-collab-project.git

cd student-collab-project

# Example Python file

echo "print('Hello from teammate!')" > hello.py

git add hello.py

git commit -m "Added hello.py"

git push origin main

**C. To Pull Classmate's Changes:**

git pull origin main

This ensures everyone has the latest code changes.

**13. Classify Regularly Used Software**

**🧠 Software Types and Examples**

| **Category** | **Examples** | **Purpose** |
| --- | --- | --- |
| **System Software** | Windows, macOS, Linux, Device Drivers | Controls hardware and manages system resources |
| **Application Software** | Chrome, MS Word, VLC, Excel, Photoshop | Performs tasks for end-users like editing, browsing |
| **Utility Software** | Antivirus (Avast), WinRAR, CCleaner, Disk Defragmenter | Enhances system performance, maintenance, and security |

**💡 Real-Life Examples**

* **System**: Windows (OS), NVIDIA Drivers
* **Application**: Chrome (Web), Excel (Spreadsheets), Zoom (Video Calls)
* **Utility**: WinRAR (Compression), Avast (Antivirus), Disk Cleanup

**14. Practice Cloning, Branching, and Merging with Git**

**🛠️ Step-by-Step Git Tutorial**

**Step 1: Create a New Repository on GitHub**

* Repo name: git-practice
* Add README.md
* Click Create

**Step 2: Clone Repository Locally**

git clone https://github.com/your-username/git-practice.git

cd git-practice

**Step 3: Create and Switch to a New Branch**

# One-step command

git checkout -b feature-hello

**Step 4: Make and Commit Changes**

echo "print('Hello from a new branch!')" > hello.py

git add hello.py

git commit -m "Add hello.py file"

**Step 5: Merge to Main**

git checkout main

git merge feature-hello

**Step 6: Push Changes**

git push origin main

You’ll now see the merged file (hello.py) in your GitHub repository.

**15. Report on Types of Application Software and Their Productivity Benefits**

**🧾 1. Introduction**

Application software helps users perform tasks such as writing, calculating, designing, or communicating. It improves individual and organizational productivity.

**💻 2. Common Types and Their Uses**

| **Type** | **Examples** | **Productivity Benefits** |
| --- | --- | --- |
| Word Processors | MS Word, Google Docs | Fast document creation, formatting, cloud collaboration |
| Spreadsheets | Excel, Google Sheets | Automated calculations, data analysis, visual charts |
| Presentation Software | PowerPoint, Google Slides | Effective idea communication via slides, teamwork collaboration |
| Database Management | MySQL, Oracle | Organized data access, querying, reporting |
| Communication Software | Zoom, Slack, Teams | Instant messaging, video conferencing, file sharing |
| Multimedia Software | VLC, Photoshop, Canva | Media playback, photo editing, content creation |
| Project Management | Trello, Asana, ClickUp | Task tracking, team coordination, deadline management |
| Web Browsers | Chrome, Firefox | Research, remote access, integration with tools and extensions |

**🏁 3. Conclusion**

Application software is vital in today's digital environment. It enables efficiency, enhances collaboration, and supports creative and data-driven tasks. Each category plays a distinct role in improving productivity for both individuals and businesses.

Here’s the elaborated version of your answers for Questions 16 to 20, expanded for clarity, academic value, and documentation purposes:

**16. Create a Flowchart Representing the Software Development Life Cycle (SDLC)**

**Definition:**  
The Software Development Life Cycle (SDLC) is a process followed by software engineers and project teams to design, develop, test, and deploy software efficiently. It ensures that software is delivered in a structured and phased manner.

**📊 Flowchart Steps:**

+----------------------+

| 1. Requirement |

| Analysis |

+----------------------+

|

v

+----------------------+

| 2. System Design |

| - Architecture |

| - UI/Database Specs |

+----------------------+

|

v

+----------------------+

| 3. Implementation |

| - Actual Coding |

+----------------------+

|

v

+----------------------+

| 4. Testing |

| - Unit/Integration |

| - Bug Fixing |

+----------------------+

|

v

+----------------------+

| 5. Deployment |

| - Release to Users |

+----------------------+

|

v

+----------------------+

| 6. Maintenance |

| - Updates & Fixes |

+----------------------+

**Explanation:**

* **Requirement Analysis**: Identify stakeholder expectations and gather exact system requirements.
* **Design**: Plan UI layout, define architecture, and determine database structure.
* **Implementation**: Translate designs into actual source code.
* **Testing**: Find and fix bugs. Perform functional and non-functional testing.
* **Deployment**: Deliver the software to the live environment.
* **Maintenance**: Ongoing support, patching, and performance upgrades.

**17. Write a Requirement Specification for a Simple Library Management System**

**📘 Software Requirements Specification (SRS)**

**1. Introduction**

* **Purpose**: To create an automated system to manage books, users, and transactions.
* **Scope**: Library staff will be able to manage books and user memberships; users will search, borrow, and return books.

**2. Functional Requirements**

* **User Authentication**: Secure login for admin, staff, and members.
* **Book Management**: Add, update, search, or delete book records.
* **Member Management**: Register members and maintain borrowing history.
* **Transaction Management**: Issue/return books, fine calculation for late returns.
* **Reports**: Generate issuance statistics, overdue reports, and book availability.

**3. Non-Functional Requirements**

* **Security**: Encrypted passwords, role-based access.
* **Performance**: Should handle up to 100 users; < 2 seconds response time.
* **Usability**: Easy UI for staff and members.
* **Availability**: Uptime of 99% with backup features.

**4. Assumptions**

* The system will be accessible via a web interface on desktop.
* Internet and browser required.

**5. Tech Stack**

* **Frontend**: HTML/CSS/JavaScript
* **Backend**: Node.js or Django
* **Database**: MySQL or PostgreSQL

**18. Perform a Functional Analysis for an Online Shopping System**

**🛒 Functional Breakdown**

**1. User Management**

* Registration, login/logout
* Profile management

**2. Product Catalog**

* View, search, filter products
* Product detail pages

**3. Cart and Checkout**

* Add to cart, modify quantity
* Checkout with shipping and payment

**4. Order Handling**

* Order placement and tracking
* Order history and cancellation

**5. Admin Functions**

* Manage products, categories
* View and process orders
* Monitor user activity

**6. Supporting Features**

* Notifications via email/SMS
* Promo codes and discounts
* Wishlist and reviews

**Functional Flow:**

User Registers/Login --> Browse Products --> Add to Cart --> Checkout -->

Order Placed --> Payment Processed --> Order Delivered --> Review Submitted

**19. Design a Basic System Architecture for a Food Delivery App**

**🍔 High-Level Architecture**

**Actors**:

* Customer
* Restaurant
* Delivery Partner
* Admin

**A. Frontend (Client Interface):**

* **Customer App**: View restaurants, place orders, track delivery
* **Restaurant Dashboard**: Accept orders, update status
* **Delivery Agent App**: Accept deliveries, GPS tracking
* **Admin Panel**: Platform management

**B. Backend (Microservices-based):**

* **Authentication Service**: Handles login, tokens
* **Order Service**: Order placement, tracking
* **Restaurant Service**: Menu and availability
* **Delivery Service**: Assign and track riders
* **Payment Service**: Integrate Razorpay, Stripe, etc.
* **Notification Service**: Email, SMS, push alerts

**C. Database Layer:**

* **Relational DB**: Orders, users, payments (PostgreSQL/MySQL)
* **NoSQL**: Menus, reviews, unstructured data (MongoDB)
* **Caching**: Redis/Memcached
* **Media Storage**: AWS S3 for food images

**D. External APIs:**

* Google Maps for location
* Payment APIs
* Notification systems

+------------+ +---------------+ +-------------+

| Customer | --> | API Gateway | --> | Microservices|

| App | +---------------+ | (Orders, etc)|

+------------+ +--+-------------+

|

+-----------v----------+

| Database & Cache |

+----------------------+

**20. Develop Test Cases for a Simple Calculator Program**

**✅ Calculator Test Case Table**

| **Test ID** | **Description** | **Input** | **Expected Output** | **Remarks** |
| --- | --- | --- | --- | --- |
| TC\_001 | Add two positive numbers | 5 + 3 | 8 | Basic test |
| TC\_002 | Add a negative and a positive number | -5 + 3 | -2 | Handles negatives |
| TC\_003 | Subtraction | 10 - 4 | 6 | Basic test |
| TC\_004 | Subtract larger from smaller | 4 - 10 | -6 | Should return negative |
| TC\_005 | Multiplication | 7 \* 6 | 42 | Basic multiplication |
| TC\_006 | Multiplication with zero | 0 \* 9 | 0 | Edge case |
| TC\_007 | Division | 20 / 4 | 5 | Basic division |
| TC\_008 | Division by zero | 9 / 0 | Error | Must handle divide-by-zero |
| TC\_009 | Float division | 5 / 2 | 2.5 | Floating-point precision |
| TC\_010 | Add floats | 1.2 + 3.4 | 4.6 | Precision check |
| TC\_011 | Invalid input | 3 + a | Error | Validate input types |
| TC\_012 | Empty input | "" | Error | Should not crash |
| TC\_013 | Complex expression (2 + 3 \* 4) | 2 + 3 \* 4 | 14 or Error | Based on precedence support |
| TC\_014 | Whitespace | 4 + 5 | 9 | Should ignore spaces |
| TC\_015 | Negative \* Negative | -3 \* -2 | 6 | Sign handling |
| TC\_016 | Chained operations | 5 + 2 - 1 | 6 | Evaluate left-to-right or via parser |

Great! Here's an **alternate method and elaborated explanation** for each of the assignment questions (21 to 24), including different code or approaches where applicable:

**21. Real-World Software Maintenance Case (Alternate Example)**

**Case Study: WhatsApp Outage Fix – October 2021**

**Issue:**  
WhatsApp experienced a global outage due to a configuration error in its backbone routers. The impact was critical—billions of users were unable to send or receive messages for hours.

**Root Cause:**  
A faulty configuration update disrupted communication between WhatsApp's data centers, breaking the network routing system.

**Maintenance Actions Taken:**

* **Emergency Rollback:** Engineers had to revert the configuration to its previous state.
* **Traffic Rerouting:** Data center engineers manually rerouted traffic to restore internal services.
* **Monitoring Upgrade:** Post-recovery, Facebook (Meta) enhanced their monitoring tools to detect global failures more rapidly.

**Outcome:**

* **Restored Services:** WhatsApp resumed normal service in ~6 hours.
* **User Trust:** Clear communication and rapid action helped regain user trust.
* **Lesson:** Critical maintenance requires rollback plans, real-time diagnostics, and continuous testing for large-scale systems.

**22. DFD – Food Delivery App (Alternate Approach)**

**Level 0 Context Diagram**

+------------+ +------------------+

| Customer | | Delivery App |

+------------+ +------------------+

\ /

\ /

v v

+----------------------+

| Food Delivery System|

+----------------------+

/ | \

/ | \

+-------------+ +-------------+ +-----------------+

| Restaurant | | Payment API | | Admin Backend |

+-------------+ +-------------+ +-----------------+

**Level 1 DFD**

1.0 Browse Menu ---> Menu DB

2.0 Place Order ---> Order DB

3.0 Assign Delivery ---> Delivery Module

4.0 Process Payment ---> Payment Gateway

5.0 Track Order ---> Notification Service

Each process is tied to a database or external API and interacts via defined data flows (arrows). Tools: draw.io, Lucidchart.

**23. DFD – Hospital Management System (Alternate Layout)**

**Level 0 Context Diagram**

+-----------+ +-----------+ +----------+

| Patient | | Doctor | | Admin |

+-----------+ +-----------+ +----------+

\ | /

\ | /

v v v

+-----------------------------+

| Hospital Management System |

+-----------------------------+

**Level 1 Processes**

1.0 Register/Login Patient --> Patient DB

2.0 Manage Appointments --> Schedule DB

3.0 Conduct Consultation --> Medical Records

4.0 Lab Test Request/Report --> Lab DB

5.0 Billing & Payment --> Billing DB

6.0 Pharmacy Dispensing --> Prescription DB

This modular design improves scalability, debugging, and data separation.

**24. Alternate Desktop Calculator GUI – Using Java (Swing)**

Instead of Python, here's the same functionality using **Java Swing**:

**Java Code – CalculatorApp.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class CalculatorApp {

public static void main(String[] args) {

JFrame frame = new JFrame("Calculator");

JTextField input = new JTextField();

JPanel panel = new JPanel(new GridLayout(5, 4));

String[] buttons = {

"7", "8", "9", "/",

"4", "5", "6", "\*",

"1", "2", "3", "-",

"0", ".", "=", "+",

"C"

};

for (String label : buttons) {

JButton button = new JButton(label);

button.setFont(new Font("Arial", Font.BOLD, 18));

panel.add(button);

button.addActionListener(e -> {

String cmd = e.getActionCommand();

if (cmd.equals("C")) {

input.setText("");

} else if (cmd.equals("=")) {

try {

input.setText(Double.toString(eval(input.getText())));

} catch (Exception ex) {

input.setText("Error");

}

} else {

input.setText(input.getText() + cmd);

}

});

}

frame.setLayout(new BorderLayout());

frame.add(input, BorderLayout.NORTH);

frame.add(panel, BorderLayout.CENTER);

frame.setSize(300, 400);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

// Simple evaluator (supports +, -, \*, /)

public static double eval(String expression) {

return new javax.script.ScriptEngineManager()

.getEngineByName("JavaScript")

.eval(expression) instanceof Double

? (double) new javax.script.ScriptEngineManager()

.getEngineByName("JavaScript")

.eval(expression)

: 0;

}

}

**How to Run:**

1. Save it as CalculatorApp.java
2. Compile: javac CalculatorApp.java
3. Run: java CalculatorApp

**Advantages Over Python:**

* Standalone compiled desktop app
* Easy to distribute via .jar
* Uses standard Java libraries, no dependencies

**Q25. Draw a flowchart representing the logic of a basic online registration system.**

**Answer:**  
The flowchart for a basic online registration system includes these main steps:

1. **Start**
2. **Input user details** (name, email, password)
3. **Validate inputs**
   * If invalid → Show error
4. **Check if email already exists**
   * If yes → Show email-in-use error
   * If no → Save user details to database
5. **Send confirmation email (optional)**
6. **Show success message**
7. **End**

**Step-by-Step Instructions:**

1. **Open Excel**
2. **Go to the "Insert" tab → Click "Shapes"**
3. Use the following shapes:
   * **Oval** for Start and End
   * **Rectangle** for processes like "Enter User Details" or "Store Info in Database"
   * **Diamond** for decisions like "Validate Input?" or "Email Exists?"
   * **Arrows** to connect the flow

**Example Flow:**

| **Shape** | **Text Inside** |
| --- | --- |
| **Oval** | Start Registration |
| **Rectangle** | Enter User Details (name, email, password) |
| **Diamond** | Validate Input Fields? |
| **Rectangle** (if No) | Show Error: Invalid Inputs |
| **Diamond** | Check if Email Already Exists? |
| **Rectangle** (if Yes) | Show Error: Email In Use |
| **Rectangle** (if No) | Store User Info in Database |
| **Rectangle** | Send Confirmation Email (optional) |
| **Rectangle** | Show Success Message |
| **Oval** | End |

**Example Layout in Excel:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | +------------------+ | |  |
|  | | Start | | |  |
|  | +------------------+ | |  |
|  | | |  |  |
|  | v |  |  |
|  | +------------------+ | |  |
|  | | Enter Details | | |  |
|  | +------------------+ | |  |
|  | | |  |  |
|  | v |  |  |
|  | +-----------------------+ | | |
|  | | Validate Inputs? | | | |
|  | +-----------------------+ | | |
|  | | | | |  |
|  | | v | |  |
|  | | +-------------------+ | | |
|  | | | Invalid? | | | |
|  | | +-------------------+ | | |
|  | | | | |  |
|  | | v | |  |
|  | | +-------------------+ | | |
|  | | | Show Error Msg | | | |
|  | | +-------------------+ | | |
|  | | | | |  |
|  | | v | |  |
|  | | +--------+ | |  |
|  | | | End | | |  |
|  | | +--------+ | |  |
|  | v |  |  |
|  | +---------------------------+ | | |
|  | | Check Email Exists? | | | |
|  | +---------------------------+ | | |
|  | | | | |  |
|  | | v | |  |
|  | | +------------------+ | | |
|  | | | Email Exists? | | | |
|  | | +------------------+ | | |
|  | | | | |  |
|  | | v | |  |
|  | | +------------------+ | | |
|  | | | Show Error Msg | | | |
|  | | +------------------+ | | |
|  | | | | |  |
|  | | v | |  |
|  | | +--------+ | |  |
|  | | | End | | |  |
|  | | +--------+ | |  |
|  | v |  |  |
|  | +----------------------+ | |  |
|  | | Store in Database | | | |
|  | +----------------------+ | |  |
|  | | |  |  |
|  | v |  |  |
|  | +----------------------+ | |  |
|  | | Send Email | | |  |
|  | +----------------------+ | |  |
|  | | |  |  |
|  | v |  |  |
|  | +----------------------+ | |  |
|  | | Show Success Message | | | |
|  | +----------------------+ | |  |
|  | | |  |  |
|  | v |  |  |
|  | +--------+ | |  |
|  | | End | | |  |
|  | +--------+ | |  |