**LEB EXERCISE:**

**Module 3: Overview of It Industry**

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

* C:

#include<stdio.h>

Int main(){

Printf(“hello world”);

Return 0;

}

* HTML:

<html>

<head>

<title>Hello</title>

</head>

<body>

<h1>Hello World</h1>

</body>

</html>

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

* User requests a website (e.g., type URL).
* DNS finds the server’s IP address.
* Browser connects to server using TCP (and HTTPS if secure).
* Data is broken into packets and sent through routers on the internet.
* Server gets request, sends response back in packets.
* Browser reassembles packets and shows the webpage.

[ Client (Browser) ]

|

v

[ DNS Server ]

|

v

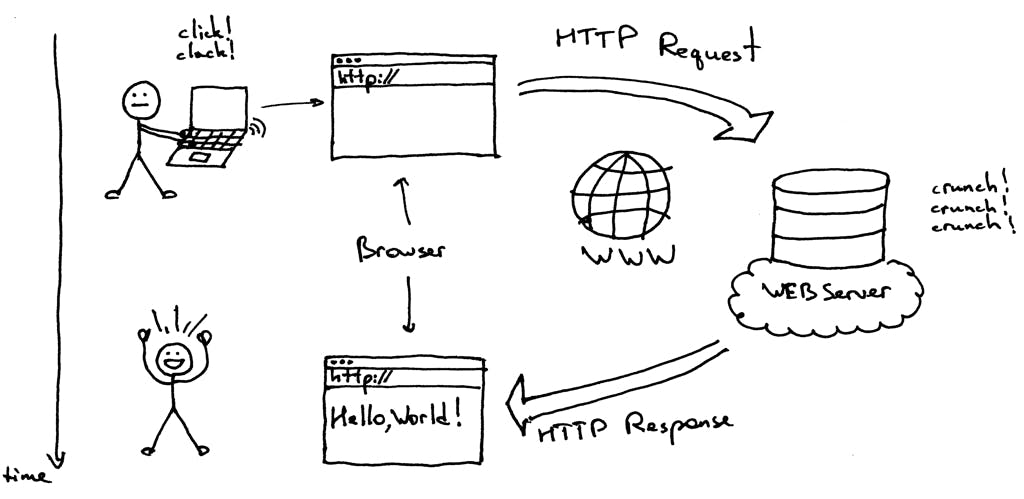
[ Internet Routers ]

|

v

[ Server (Website) ]

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



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

* Broadband (DSL/Cable)
* Pros: Widely available, affordable, good for general use
* Cons: Slower than fiber, speed may drop during peak times
* Fiber Optic
* Pros: Very fast, reliable, great for heavy use
* Cons: Expensive, limited to cities or towns
* Satellite
* Pros: Works in remote/rural areas
* Cons: Slow, affected by weather, high delay
* Mobile Data (3G/4G/5G)
* Pros: Portable, fast with 4G/5G, easy to set up
* Cons: Limited data, speed depends on signal
* Fixed Wireless
* Pros: Good for rural areas, better than satellite
* Cons: Needs clear signal, affected by weather

1. imulate HTTP and FTP requests using command line tools (e.g., curl).

* HTTPS Request(GET):

curl <http://example.com>

* HTTPS Request(POST):

curl -X POST -d "name=harsh&age=25" <http://example.com/form>

* FTP Request (Download a file):

curl -u usernam:password <ftp://ftp.example.com/file.txt>

* FTP Request (Upload a file):

curl -T file.txt -u username:password <ftp://ftp.example.com/>

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

* SQL Injection
* What it is: Attacker inserts malicious SQL code into input fields to access or manipulate the database.
* Solution:
* Use prepared statements or parameterized queries
* Validate and sanitize all user inputs
* Cross-Site Scripting (XSS)
* What it is: Attacker injects malicious scripts into web pages viewed by others.
* Solution:
* Escape HTML output
* Use Content Security Policy (CSP)
* Validate user input
* Broken Authentication
* What it is: Weak login systems allow attackers to bypass or steal user identities.
* Solution:
* Use strong password policies
* Implement multi-factor authentication (MFA)
* Secure session management (timeouts, tokens)

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

* Google Chrome: Application Software (Web Browser)
* Microsoft Word: Application Software (Productivity/Word Processor)
* WhatsApp: Application Software (Communication)
* Android OS / iOS: System Software (Operating System)
* Windows Defender / macOS Security: System Software (Utility/Antivirus built into OS)

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

* Presentation Tier (Client):
* Purpose: User interface (UI)
* Example: Browser, mobile app
* Technology: HTML, CSS, JavaScript, React
* Application Tier (Server):
* Purpose: Business logic
* Example: Processes user requests
* Technology: Node.js, Java, Python, PHP
* Data Tier (Database):
* Purpose: Stores and retrieves data
* Example: User info, products, etc.
* Technology: MySQL, MongoDB, PostgreSQL

[Client/Browser] <-- Presentation Tier

|

[Web/Application Server] <-- Application Tier

|

[Database Server] <-- Data Tier

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

* Presentation Layer (UI)
* Function: Interacts with users.
* Example:
* User views product listings.
* Adds items to cart, enters payment info.
* Technology: HTML, CSS, JavaScript, React
* Business Logic Layer
* Function: Handles core rules and processing.
* Example:
* Validates user login.
* Calculates total price, applies discounts.
* Manages order process.
* Technology: Java, Node.js, Python (Back-end server)
* Data Access Layer
* Function: Manages communication with the database.
* Example:
* Fetches product details.
* Stores order information.
* Updates stock levels.
* Technology: SQL, ORM tools (like Sequelize, Hibernate)

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

* Development Environment
* Used by developers to write and test code.
* Includes tools like code editors, debuggers.
* Example: localhost with Node.js and MySQL.
* Testing Environment
* Used to test the application before release.
* Includes test data, QA tools.
* Ensures app works as expected.
* Production Environment
* Live environment accessed by users.
* Must be stable, secure, and optimized.

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

* <!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Hello World</title>

</head>

<body>

<h1>Hello, World!</h1>

</body></html>

* Create a new GitHub repository
* Log in to GitHub
* Click on “New” (or “+” → “New repository”)
* Give it a Repository name — e.g., hello-world-html
* Add a description if you want (optional)
* Keep it Public
* DO NOT initialize with a README (optional)
* Click Create repository
* Upload your file
* After creating the repo, click “Upload files”
* Drag and drop your hello.html file into the upload area

or click “choose your files”

* Scroll down and click “Commit changes”

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

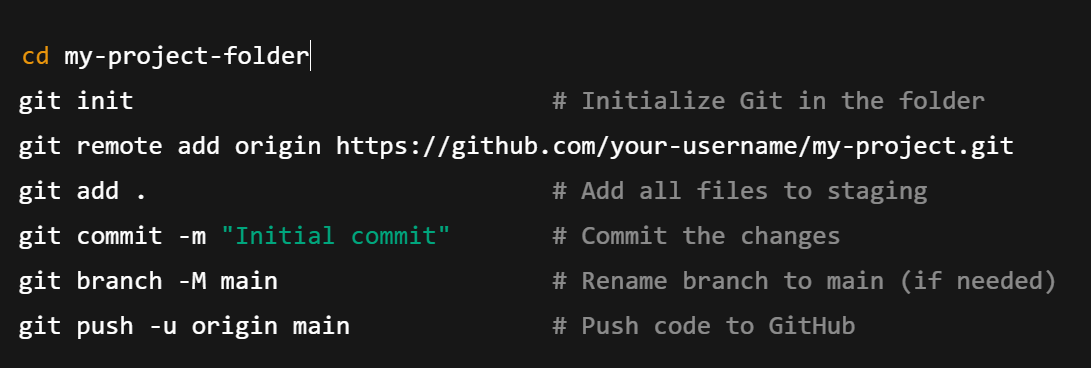
* Create a GitHub Repository
* Go to https://github.com
* Click "New" (top-left or via + icon)
* Enter:

Repository Name (e.g., my-project)

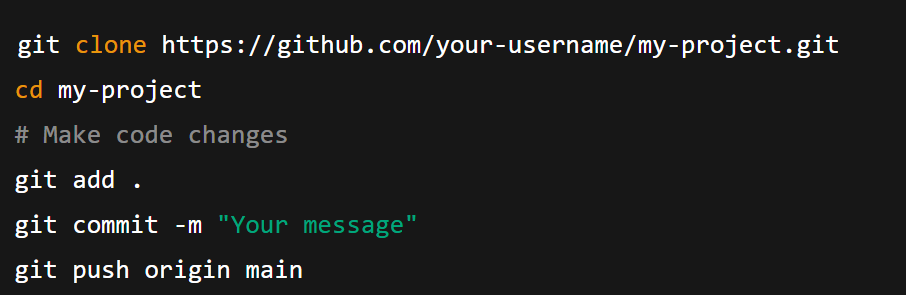
Optional: Description

Choose Public or Private

* Click "Create repository"
* If You Have a Local Project:



* If You Cloned a Repo:



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

The following is a list of commonly used software classified by category:

* System Software:

Operating System: Windows, macOS, Linux (e.g., Ubuntu)

Device Drivers: Graphics card drivers, printer drivers, audio drivers

* Application Software:

Web Browsers: Google Chrome, Mozilla Firefox, Microsoft Edge

Productivity Suites: Microsoft Office (Word, Excel, PowerPoint), GoogleWorkspace (Docs, Sheets, Slides)

Communication Software: Zoom, Microsoft Teams, Slack

Multimedia Software: VLC Media Player, Spotify, Adobe Photoshop, GIMP

Development Tools: Visual Studio Code, PyCharm, Git

* Utility Software:

Antivirus/Security Software: Windows Defender, Avast, Malwarebytes

File Management Tools: File Explorer (Windows), Finder (macOS), WinRAR, 7-Zip

Disk Management Tools: Disk Cleanup, Disk Defragmenter (Windows)

Backup Software: Veeam Backup & Replication, Acronis True Image.

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

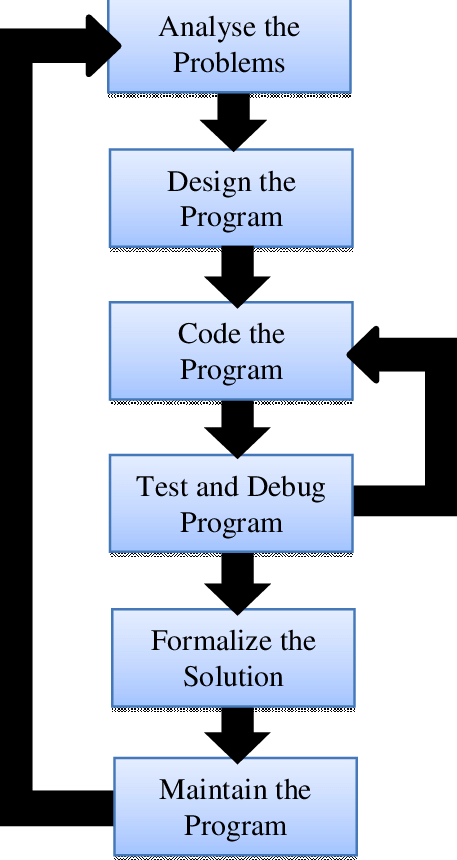
* Initialize a Local Repository:
* Create a new directory for your project:
* Initialize a Git repository within this directory:
* Create and Commit Files:
* Commit the changes to the repository.
* Branching:
* Create a new branch for a feature or bug fix (e.g., feature-branch):
* Make changes on this new branch (e.g., add content to README.md):
* Commit the changes on feature-branch.
* Merging:
* Switch back to the main branch.
* Merge the changes from feature-branch into main:
* Cloning (Simulated Remote Repository):
* Navigate to a different location outside your current project directory
* Clone your existing local repository as if it were a remote one:
* Navigate into the cloned repository.

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

* Productivity Software:
* Business Software:
* Communication and collaboration software:
* Project management software:
* Graphic Design Software:
* How Application Software Improves Productivity:
* Application software automates repetitive tasks, freeing up time for more
* complex and strategic work.
* Software helps organize, store, and analyse data, leading to better

insights and informed decisions.

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



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

* Purpose
* To manage library books, members, issuing, and returns efficiently.
* Functional Requirements
* Add/update/delete books and members
* Search books by title or author
* Issue and return books
* Track overdue books and calculate fines
* Admin login for system control
* Non-Functional Requirements
* User-friendly interface
* Secure login system
* Fast performance
* Reliable data storage
* Users
* Admin: Manages books and members
* User/Student: Searches and borrows books

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

* User Functions
* Register/Login: Users can create an account and log in securely.
* Browse Products: Users can view product categories and details.
* Search: Users can search for products by name, category, or brand.
* Add to Cart: Users can add selected items to the cart.
* Place Order: Users can enter shipping details and place an order.
* Payment: Users can pay using credit card, UPI, or other options.
* Order Tracking: Users can view order status (shipped, delivered).
* Admin Functions
* Manage Products: Add, update, or remove products.
* Manage Orders: View, approve, or cancel customer orders.
* Manage Users: View and manage customer accounts.
* Reports: View sales, inventory, and customer activity reports.
* System Functions
* Authentication: Secure login for users and admins.
* Inventory Update: Automatically reduces stock when items are sold.
* Notifications: Send email/SMS for order confirmation and delivery.
* Data Storage: Save product, order, and user data in a database.

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

* Client Side (Frontend)
* User App: Browse restaurants, place orders, track delivery
* Delivery App: View delivery tasks, update status
* Admin Panel: Manage restaurants, orders, users
* Server Side (Backend)
* Handles:
  + User authentication
  + Order processing
  + Restaurant & menu management
  + Payment processing
  + Notifications (SMS/Email)
* Database
* Stores:
  + User info
  + Orders
  + Menus
  + Delivery status
  + Payments
* Third-Party Services
* Payment Gateway: e.g., Razorpay, Stripe
* Maps API: For location & route tracking
* Notification Service: For order and delivery updates

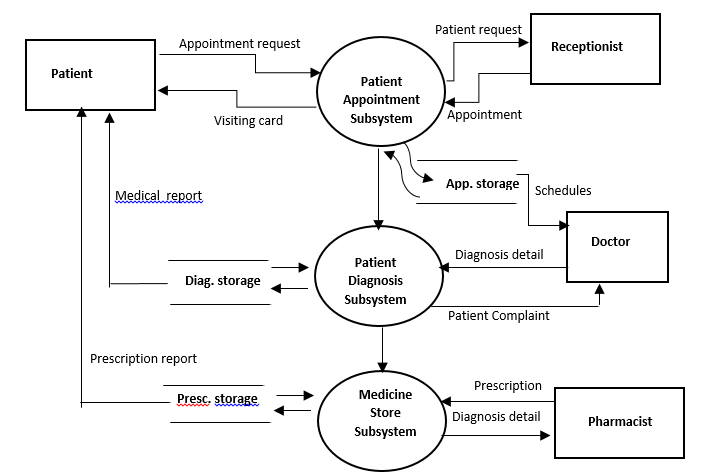
1. Develop test cases for a simple calculator program.

| **Test Case ID** | **Test Description** | **Input** | **Expected Output** | **Status** |
| --- | --- | --- | --- | --- |
| TC01 | Test Addition | 5 + 3 | 8 | Pass/Fail |
| TC02 | Test Subtraction | 10 - 4 | 6 | Pass/Fail |
| TC03 | Test Multiplication | 6 \* 7 | 42 | Pass/Fail |
| TC04 | Test Division | 20 / 5 | 4 | Pass/Fail |
| TC05 | Divide by Zero | 8 / 0 | Error or Exception | Pass/Fail |
| TC06 | Invalid Input Characters | "a" + 3 | Error Message | Pass/Fail |
| TC07 | Negative Number Addition | -5 + (-3) | -8 | Pass/Fail |
| TC08 | Decimal Addition | 2.5 + 3.2 | 5.7 | Pass/Fail |
| TC09 | Multiple Operations | 2 + 3 \* 4 | 14 (if BODMAS) | Pass/Fail |
| TC10 | Clear or Reset Function | Clear Input | 0 or Empty Screen | Pass/Fail |
|  |  |  |  |  |

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

* Background
* During the COVID-19 pandemic, Zoom saw a massive surge in users worldwide. However, by early 2020, several security and privacy issues were discovered in the application.
* Problem
* Zoombombing: Uninvited users disrupted meetings.
* Weak encryption: Zoom used AES-128 ECB mode (not secure).
* User data routing: Some data routed through China, causing privacy concerns.
* No end-to-end encryption for most meetings.
* Maintenance Actions Taken
* Zoom paused new feature development for 90 days to focus on security.
* Released multiple patches and updates:
* Implemented AES-256 GCM encryption
* Added waiting rooms, meeting passwords, and host controls
* Improved UI for privacy settings
* Introduced Zoom 5.0 with major security upgrades.
* Outcome
* Restored user trust and reduced disruptions.
* Improved brand image.
* Continued growth with better user experience and safety.

1. Create a DFD for a hospital management system.



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

A simple desktop calculator application can be built using a GUI library like Python's Tkinter.

**Steps to Build a Simple Calculator with Tkinter:**

* **Import Tkinter:** Begin by importing the tkinter module.
* **Create the Main Window:** initialize the main application window.
* **Create an Entry Field for Display:** Add an Entry widget to display input and results.
* **Define Button Functionality:**

Create functions to handle button clicks (e.g., adding numbers, performing operations, clearing the display).

* + **button\_click(number):** Appends the clicked number to the display.
  + **button\_clear():** Clears the display.
  + **button\_add(), button\_subtract(), button\_multiply(), button\_divide():** Store the current number and the operation.
  + **button\_equal():** Evaluates the expression and displays the result.
* **Create and Place Buttons:**

Create Button widgets for numbers (0-9), operations (+, -, \\*, /), equals, and clear. Arrange them using grid() for a calculator-like layout.

* **Run the Main Loop:** Start the Tkinter event loop to make the window interactive.

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

