Module 1 - Overview of IT Industry - Theory Exercises

1. What is a Program?

A program is a collection of instructions written in a programming language that tells a computer how to perform a specific task. These instructions define the logic and steps needed to process inputs and produce outputs.
For example, a calculator program takes numbers from the user, applies mathematical operations (addition, subtraction, etc.), and displays the result.
Without programs, a computer is just a piece of hardware — programs bring it to life and make it useful.

2. What is Programming?

Programming is the process of designing and building a set of instructions that a computer can execute. It involves problem-solving, creating algorithms, and writing code using programming languages like Python, Java, or C.
The purpose of programming is to translate human ideas into a form that a machine can understand and act upon. For example, creating a mobile app or a website both require programming.

3. Key Steps in the Programming Process

The programming process is generally divided into the following stages:

1. Problem Definition – Clearly understanding the problem to be solved.
2. Planning C Design – Choosing the right approach, writing pseudo code, or creating flowcharts.
3. Coding – Writing the actual program using a programming language.
4. Testing C Debugging – Running the program to find and fix errors.
5. Deployment – Making the program available for users.
6. Maintenance – Updating the program to fix issues or add new features over time. Following these steps ensures that the final program is reliable and meets user requirements.

4. **Differences Between High-Level and Low-Level Languages**

Point	High-Level Languages	Low-Level Languages
Definition	Closer to human language, easy to understand and write.	Closer to machine language, harder to read and write.
Readability	Easy for humans to read and debug.	Difficult for humans to understand.
Execution Speed	Slower due to translation process.	Very fast and efficient.
Portability	Can run on multiple platforms with minimal changes.	Platform-dependent.
Examples	Python, Java, C++.	Assembly, Machine code.

5. Roles of the Client and Server in Web Communication

- Client – The device or application (like a browser) that requests services or data from the server. It displays information to the user and sends user actions (clicks, forms) to the server.
- Server – A powerful computer or program that processes client requests, retrieves or processes data, and sends a response back.
  Example: When you visit Google, your browser (client) requests a search page, and Google's server responds with the search results.

6. Function of the TCP/IP Model and Its Layers

The TCP/IP model describes how data moves across a network.
Layers:

1. Application Layer – Interfaces for communication (e.g., HTTP for websites, FTP for file transfer).
2. Transport Layer – Ensures reliable delivery of data (TCP) or fast, connection less transfer (UDP).
3. Internet Layer – Routes packets across networks using IP addresses.
4. Network Access Layer – Handles physical network connections (Ethernet, Wi-Fi).
   This model ensures that devices on different networks can communicate effectively.

7. Client-Server Communication

In client-server communication, the client initiates a request, and the server processes it and sends back a response.
Example: When you open YouTube, your device (client) requests video data, and YouTube's server sends it back for playback. This interaction happens using defined network protocols like HTTP.

8. Broadband vs Fiber-Optic Internet

Point	**Broadband Internet**	**Fiber-Optic Internet**
1. **Technology**	Uses copper cables (DSL or coaxial).	Uses thin strands of glass or plastic to transmit data as light.
2. **Speed**	Slower compared to fiber, usually up to 100 Mbps.	Extremely fast, can reach 1 Gbps or more.
3. **Reliability**	More prone to interference and signal loss over distance.	Highly reliable with minimal signal loss.
4. **Cost**	Generally cheaper and widely available.	More expensive and limited to certain areas.

9. HTTP vs HTTPS

• Point	• HTTP	• HTTPS
• Security	• Data is sent without encryption.	• Data is encrypted using SSL/TLS.
• Privacy	• Vulnerable to eavesdropping.	• Protects against interception and tampering.
• Trust	• No padlock icon in browser address bar.	• Shows padlock icon for secure connection.

10. Role of Encryption in Securing Applications

Encryption transforms readable data into an unreadable format using algorithms and keys.
Its main roles are:

- Protecting sensitive information like passwords and credit card details.
- Ensuring that even if data is intercepted, it cannot be understood without the decryption key.
- Maintaining trust between users and applications.

11. System Software vs Application Software

Point	System Software	Application Software
Purpose	Manages hardware and provides platform for applications.	Helps users perform specific tasks.
Dependency	Can run independently.	Requires system software to function.
Examples	Operating systems, device drivers.	MS Word, browsers, games.

12. Significance of Modularity in Software Architecture

Modularity means dividing software into smaller, manageable components (modules).
Benefits include:

- Easier debugging and maintenance.
- Reusability of modules in other projects.
- Parallel development by multiple teams.
- Improved readability and organization.

13. Importance of Layers in Software Architecture

Layers organize a system into different functional parts, such as:

- Presentation Layer – User interface.
- Business Logic Layer – Application logic and rules.

- Data Access Layer – Communication with the database.
  This separation increases maintainability, scalability, and flexibility.

14. **Importance of a Development Environment**

A development environment is a setup that includes tools like code editors, compilers, libraries, and debugging utilities.
It allows developers to build and test applications in a controlled space before deploying to production, reducing errors and downtime.

15. Source Code vs Machine Code

Point	Source Code	Machine Code
Format	Written in human-readable language.	Written in binary (0s and 1s).
Understandable By	Programmers.	Computer CPU.
Conversion	Needs compilation or interpretation.	Directly executed by CPU.
Example	printf("Hello");	10110000 01100001

16. **Importance of Version Control**

Version control tracks changes to files and allows developers to collaborate without overwriting each other’s work.
Benefits:

- Undoing mistakes by reverting to previous versions.
- Working on separate features using branches.
- Keeping a complete history of changes.

17. **Benefits of Using GitHub for Students**

- Free hosting for projects.
- Collaboration with classmates.
- Building a public portfolio for future employers.
- Exposure to industry-standard tools and workflows.

18. Open-source vs Proprietary Software

Point	Open-Source Software	Proprietary Software
Accessibility	Source code is freely available.	Source code is closed and restricted.
Cost	Usually free.	Usually paid.
Modification	Can be modified by anyone.	Cannot be modified without permission.
Examples	Linux, GIMP.	Windows, MS Office.

19. How Git Improves Collaboration

Git allows multiple contributors to work on the same codebase simultaneously.
It handles merging changes, tracks file history, and reduces conflicts in collaborative development.

20. Role of Application Software in Businesses

Application software automates tasks like accounting, inventory management, and customer support. It improves productivity, reduces errors, and enables better decision-making.

21. Main Stages of the Software Development Process (SDLC)

1. Requirement Analysis - Understanding user needs.
2. Design – Planning system architecture.
3. Implementation – Writing code.
4. Testing – Finding and fixing errors.
5. Deployment – Delivering to users.
6. Maintenance – Updating and improving the system.

22. Importance of Requirement Analysis

Requirement analysis prevents misunderstandings between clients and developers. It ensures the software solves the correct problem and meets expectations.

23. Role of Software Analysis

Software analysis determines what the system should do and how it should be structured. It forms the foundation for the design and implementation phases.

24. Key Elements of System Design

- Architecture – Structure of the software system.
- Data Design – How data is stored and accessed.
- Interface Design – User interaction methods.
- Security – Protecting the system from threats.

25. Importance of Software Testing

Testing ensures that software is functional, secure, and user-friendly. It helps avoid costly failures after release.

26. Types of Software Maintenance

1. Corrective – Fixing defects.
2. Adaptive – Updating to work with new systems.
3. Perfective – Enhancing features.
4. Preventive – Reducing future risks.

27. Web Application

Point	Web Applications	Desktop Applications
Location	Runs inside a web browser.	Installed locally on a device.

Point	Web Applications	Desktop Applications
Internet Requirement	Needs internet to function.	Can work offline.
Updates	Updated on server, instantly available to all users.	Requires manual updates on each device.
Accessibility	Accessible from any device with a browser.	Accessible only on the installed device.

28. Advantages of Web Applications Over Desktop Applications

- No installation required.
- Accessible from multiple devices.
- Centralized updates.

29. Role of UI/UX Design in Application Development

Point	Native Mobile Apps	Hybrid Mobile Apps
Development	Built for a specific OS using platform-specific languages.	Built using web technologies for multiple OS.
Performance	High performance, optimized for platform.	Slightly lower due to compatibility layers.
Maintenance	Requires separate code for each OS.	One codebase works for multiple platforms.
Examples	Android apps in Kotlin/Java, iOS apps in Swift.	Apps built with Flutter, React Native.

30. **Native vs Hybrid Mobile Apps**

- Native – Built for one platform, high performance, full hardware access.
- Hybrid – One codebase for multiple platforms, faster development, but may have performance limitations.

31. **Significance of DFDs in System Analysis**

Point	Desktop Applications	Web Applications
Installation	Requires installation on the system.	No installation required.
Platform Dependency	Usually works only on specific OS.	Works on multiple platforms via browser.
Performance	Often faster and can use full system resources.	May be slower, limited by browser capabilities.
Accessibility	Accessible only on installed device.	Accessible anywhere with internet.

32. **Pros and Cons of Desktop Applications**

Pros	Cons
Works offline without internet.	Requires installation on each device.
Often faster and more responsive than web apps.	Platform-dependent, may not work on all OS.
Can utilize full system resources (CPU, GPU, storage).	Updating requires manual effort on each device.
Generally offers better performance for heavy tasks (e.g., video editing, gaming).	Not accessible remotely unless additional setup is done.

33. How Flowcharts Help in Programming and System Design

Flowcharts visually represent the steps and decisions in a program, making complex logic easier to understand. They help in planning before coding, reducing errors, and improving communication among team members. Flowcharts also make debugging faster, serve as useful documentation for future maintenance, and break down complex problems into simpler steps.