**What is a Program?**

A **program** is a set of instructions written in a specific programming language that a computer can execute to perform a specific task. Programs act as intermediaries between human commands and machine actions, allowing us to control computer behavior effectively.

**LAB EXERCISE**

**Write a simple "Hello World" program in two languages:**

1. **C Language**
2. #include <stdio.h>
3. int main() {
4. printf("Hello, World!\n");
5. return 0;

}

1. **Python**

print("Hello, World!")

**Comparison:**

* **Syntax**: C requires more structure with #include and main(), while Python uses a simple print() function.
* **Semicolons**: C ends statements with semicolons; Python does not.
* **Compilation**: C is compiled, while Python is interpreted.

**What is Programming?**

Programming is the process of creating instructions (software) that tell a computer how to perform tasks. It involves problem-solving, logical thinking, and using languages like C, Python, or JavaScript.

**Key Steps in the Programming Process**

1. **Problem Analysis**
2. **Design**
3. **Coding**
4. **Testing**
5. **Maintenance**

**Types of Programming Languages**

**High-Level vs. Low-Level Languages:**

* High-level languages (Python, Java): Easier for humans to understand, machine-independent.
* Low-level languages (Assembly, Machine Code): Closer to hardware, faster, but harder to code.

**World Wide Web & How Internet Works**

**LAB EXERCISE**

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

**THEORY EXERCISE**

**Describe the roles of the client and server in web communication.**

* **Client**: Requests services (e.g., a web browser requesting a webpage).
* **Server**: Responds to client requests (e.g., delivering the webpage).

**Network Layers on Client and Server**

**LAB EXERCISE**

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

Example in Python:

**Client:**

import socket

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect(("localhost", 8080))

client\_socket.sendall(b"Hello, Server!")

response = client\_socket.recv(1024)

print("Server says:", response.decode())

client\_socket.close()

**Server:**

import socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind(("localhost", 8080))

server\_socket.listen(1)

print("Server is listening...")

conn, addr = server\_socket.accept()

print("Connected by", addr)

data = conn.recv(1024)

conn.sendall(b"Hello, Client!")

conn.close()

**THEORY EXERCISE**

**Explain the function of the TCP/IP model and its layers:**

1. **Application Layer**: Interfaces with applications.
2. **Transport Layer**: Manages end-to-end communication.
3. **Internet Layer**: Routes packets.
4. **Link Layer**: Handles physical network communication.

**Client and Servers**

**THEORY EXERCISE**

**Explain Client-Server Communication:**

* **Client**: Initiates requests for resources.
* **Server**: Processes requests and sends back responses.

**Types of Internet Connections**

**LAB EXERCISE**

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

**THEORY EXERCISE**

**How does broadband differ from fiber-optic internet?**

* **Broadband**: Uses copper cables; slower than fiber.
* **Fiber-optic**: Uses light-based transmission; faster, more reliable.

**Protocols**

**LAB EXERCISE**

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

Example:

curl http://example.com

ftp ftp://ftp.example.com

**THEORY EXERCISE**

**What are the differences between HTTP and HTTPS protocols?**

* **HTTP**: Data sent in plain text.
* **HTTPS**: Uses SSL/TLS encryption for security.

**Application Security**

**LAB EXERCISE**

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

1. **SQL Injection**: Validate inputs, use prepared statements.
2. **Cross-Site Scripting (XSS)**: Escape user inputs, use content security policies.
3. **Insecure Authentication**: Implement strong password policies and multi-factor authentication.

**THEORY EXERCISE**

**What is the role of encryption in securing applications?**

Encryption protects data by converting it into a secure format that requires a key to decrypt, ensuring confidentiality and integrity.

**1. Software Applications and Its Types**

**LAB EXERCISE**

Identify and classify 5 applications you use daily:

* **System Software**:
  1. Windows Operating System
  2. macOS
* **Application Software**:  
  3. Microsoft Word  
  4. Google Chrome  
  5. WhatsApp

**THEORY EXERCISE**

**Difference between System Software and Application Software**:

* **System Software** manages hardware and provides a platform for application software. Example: Operating systems.
* **Application Software** is designed for end-users to perform specific tasks. Example: Browsers, Word processors.

**2. Software Architecture**

**LAB EXERCISE**

Design a basic three-tier software architecture for a web application:

* **Presentation Layer**: Handles the user interface (HTML, CSS, JavaScript).
* **Business Logic Layer**: Contains logic and processing (Node.js, Python).
* **Data Access Layer**: Manages database interactions (MongoDB, MySQL).

**THEORY EXERCISE**

**Significance of Modularity in Software Architecture**:

* Enhances maintainability, scalability, and reusability of code.
* Simplifies debugging and updates.

**3. Layers in Software Architecture**

**LAB EXERCISE**

Case Study on Layers of a Software System:

* **Presentation Layer**: User interface (UI/UX components).
* **Business Logic Layer**: Core functionality (processing user requests).
* **Data Access Layer**: Data retrieval/storage.

**THEORY EXERCISE**

**Importance of Layers**:

* Promotes separation of concerns, making the system easier to manage and scale.

**4. Software Environments**

**LAB EXERCISE**

Explore environments: Set up a basic environment in a virtual machine using VirtualBox or VMware.

* **Development Environment**: For writing and testing code.
* **Testing Environment**: For quality assurance.
* **Production Environment**: Live system for end-users.

**THEORY EXERCISE**

**Importance of Development Environment**:

* Helps in writing, testing, and debugging before deployment, ensuring fewer bugs in production.

**5. Source Code**

**LAB EXERCISE**

Write and upload your first source code to GitHub. Example: A simple “Hello World” program in Python or JavaScript.

**THEORY EXERCISE**

**Difference between Source Code and Machine Code**:

* **Source Code** is human-readable code written in a programming language.
* **Machine Code** is binary code executed by the computer's CPU.

**6. GitHub and Introductions**

**LAB EXERCISE**

Create a GitHub repository and push changes:

* Initialize a repository, add a file, commit, and push it to GitHub.

**THEORY EXERCISE**

**Importance of Version Control**:

* Tracks changes, allows collaboration, and provides a history of revisions.

**7. Student Account in GitHub**

**LAB EXERCISE**

Create a GitHub student account and collaborate with a classmate on a small project.

**THEORY EXERCISE**

**Benefits of Using GitHub for Students**:

* Enhances coding skills through collaboration.
* Provides project showcases for resumes.

**8. Types of Software**

**LAB EXERCISE**

Classify software into categories:

* **System Software**: Operating System (Windows, Linux).
* **Application Software**: MS Word, Zoom.
* **Utility Software**: Antivirus, Disk Cleanup.

**THEORY EXERCISE**

**Differences between Open-Source and Proprietary Software**:

* **Open-Source Software** allows access to source code, e.g., Linux.
* **Proprietary Software** restricts access, e.g., Windows.

**1. GIT and GITHUB Training**

**LAB EXERCISE**

Follow a GIT tutorial to practice the following:

* **Cloning**: Use git clone <repository\_URL> to download a copy of a repository.
* **Branching**: Create a new branch with git branch <branch\_name> and switch to it with git checkout <branch\_name>.
* **Merging**: Merge a branch using git merge <branch\_name> while on the main branch.

**THEORY EXERCISE**

**How GIT Improves Collaboration in Software Development**:

* Provides version control to track changes.
* Supports branching for parallel development.
* Facilitates teamwork by merging code and resolving conflicts.
* Maintains a history of modifications, enhancing traceability.

**2. Application Software**

**LAB EXERCISE**

Write a report on various types of application software:

1. **Word Processors**: Microsoft Word — used for document creation.
2. **Spreadsheet Software**: Excel — for data organization and analysis.
3. **Database Software**: MySQL — for data storage and management.
4. **Web Browsers**: Chrome — for accessing internet resources.
5. **Multimedia Software**: VLC — for media playback.

**Productivity**: Application software increases efficiency by automating tasks, improving data analysis, and enhancing communication.

**THEORY EXERCISE**

**Role of Application Software in Businesses**:

* Streamlines operations with management systems.
* Enhances data analysis through business intelligence tools.
* Improves communication with email and conferencing software.

**3. Software Development Process**

**LAB EXERCISE**

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

* **Phases**: Planning → Analysis → Design → Development → Testing → Deployment → Maintenance.

**THEORY EXERCISE**

**Main Stages of Software Development**:

1. **Planning**: Define goals and scope.
2. **Requirement Analysis**: Understand user needs.
3. **Design**: Create system architecture.
4. **Development**: Write code.
5. **Testing**: Find and fix defects.
6. **Deployment**: Release the product.
7. **Maintenance**: Update and improve the software.

**4. Software Requirement**

**LAB EXERCISE**

Write a requirement specification for a Library Management System:

* **Functional Requirements**:
  1. Allow users to borrow and return books.
  2. Search books by title or author.
* **Non-functional Requirements**:
  1. The system must handle 100 concurrent users.
  2. Response time should be less than 2 seconds.

**THEORY EXERCISE**

**Importance of Requirement Analysis**:

* Prevents misunderstandings.
* Helps in planning accurate timelines and budgets.
* Acts as a reference for validation and testing.

**5. Software Analysis**

**LAB EXERCISE**

Perform a functional analysis for an Online Shopping System:

* **User Functions**:
  1. Register and log in.
  2. Browse and search products.
  3. Add items to the cart and check out.
* **Admin Functions**:
  1. Manage product inventory.
  2. Process orders.

**THEORY EXERCISE**

**Role of Software Analysis**:

* Identifies what the system should do.
* Clarifies user needs and system constraints.
* Provides a foundation for design and testing.

**6. System Design**

**LAB EXERCISE**

Design a basic system architecture for a Food Delivery App:

* **User Interface**: Mobile app/web interface for customers.
* **Business Logic**: Handles order processing and payment.
* **Database**: Stores customer, restaurant, and order information.

**THEORY EXERCISE**

**Key Elements of System Design**:

* **Data Structures**: Organize and manage data efficiently.
* **System Components**: Modules and their interactions.
* **User Interfaces**: Inputs and outputs for user interaction.
* **Security**: Protects data and user privacy.

**1. Software Testing**

**LAB EXERCISE**

**Develop Test Cases for a Simple Calculator Program**:

* Test Case 1: **Addition** — Input: 5 + 3, Expected Output: 8.
* Test Case 2: **Subtraction** — Input: 9 - 4, Expected Output: 5.
* Test Case 3: **Division by Zero** — Input: 7 ÷ 0, Expected Output: Error or Infinity.
* Test Case 4: **Multiplication** — Input: 6 × 3, Expected Output: 18.

**THEORY EXERCISE**

**Why Software Testing is Important**:

* Ensures software functionality and reliability.
* Detects bugs and prevents failures.
* Improves user satisfaction and trust.

**2. Maintenance**

**LAB EXERCISE**

**Document a Real-World Maintenance Case**:  
Example: **Microsoft Windows Updates** — Frequent security and performance patches address vulnerabilities and improve system stability.

**THEORY EXERCISE**

**Types of Software Maintenance**:

1. **Corrective Maintenance**: Fixes bugs and errors.
2. **Adaptive Maintenance**: Updates software for compatibility with changing environments.
3. **Perfective Maintenance**: Enhances performance and features.
4. **Preventive Maintenance**: Anticipates future issues to improve reliability.

**3. Development**

**THEORY EXERCISE**

**Key Differences between Web and Desktop Applications**:

* **Web Applications**: Accessed via browsers, require internet, and are platform-independent.
* **Desktop Applications**: Installed on local machines, do not require internet, and are platform-dependent.

**THEORY EXERCISE**

**Advantages of Web Applications**:

* Accessible from any device with internet.
* Easier to update and maintain centrally.
* Cross-platform compatibility.

**4. Designing**

**THEORY EXERCISE**

**Role of UI/UX in Application Development**:

* **UI (User Interface)** focuses on appearance and layout.
* **UX (User Experience)** ensures ease of use and user satisfaction.
* Good design improves usability, engagement, and efficiency.

**5. Mobile Application**

**THEORY EXERCISE**

**Differences between Native and Hybrid Mobile Apps**:

* **Native Apps**: Built for specific platforms (e.g., iOS, Android) using platform-specific languages (Swift, Kotlin).
* **Hybrid Apps**: Use web technologies (HTML, CSS, JavaScript) and work across platforms using frameworks like React Native or Flutter.

**6. Data Flow Diagram (DFD)**

**LAB EXERCISE**

Create a DFD for a Hospital Management System:

1. **Entities**: Patients, Doctors, Admin.
2. **Processes**: Appointment scheduling, Billing, Medical record management.
3. **Data Stores**: Patient records, Payment details.

**THEORY EXERCISE**

**Significance of DFDs in System Analysis**:

* Visualizes data movement and processing.
* Clarifies system functionality and user interactions.

**7. Desktop Application**

**LAB EXERCISE**

**Build a Simple Calculator with GUI**: Use libraries like **Tkinter** (Python) or **Swing** (Java).

**THEORY EXERCISE**

**Pros and Cons of Desktop Applications**:

* **Pros**: Faster performance, offline access, more control over system resources.
* **Cons**: Platform dependency, manual updates.

**8. Flow Chart**

**LAB EXERCISE**

**Draw a Flowchart for an Online Registration System**:

* Start → User Inputs Details → Validate Inputs → Check Availability → Register → Confirmation → End.

**THEORY EXERCISE**

**How Flowcharts Help**:

* Provide a visual representation of logic.
* Enhance understanding and debugging.
* Serve as a blueprint for coding.