Module 1 – Foundation

THEORY EXERCISE:

1. What is HTTP?

HTTP stands for **HyperText Transfer Protocol**.

- It is the **protocol (set of rules)** used by web browsers and servers to **communicate** with each other.
- When you type a website URL (like `https://www.google.com`) and press Enter, your browser sends an **HTTP request** to the server.
- The server responds with an **HTTP response**, which includes the website content (HTML, images, etc.).

 Example:
When you visit `http://example.com`, your browser uses HTTP to ask the server for the website content.

---

2. What is a Browser? How do they work?

A **browser** is a software application used to **access and view websites** on the Internet.

 Examples: Chrome, Firefox, Safari, Edge

✅ **How Browsers Work:**

1. **User Enters URL** – e.g., `www.google.com`
2. **DNS Lookup** – Browser finds the IP address of the website.
3. **HTTP Request** – Browser sends a request to the server.
4. **Server Responds** – Server sends back HTML, CSS, images, etc.
5. **Rendering** – Browser processes the files and **displays the web page** on your screen.

---

3. What is a Domain Name?

A **domain name** is the **website address** you type in a browser to visit a site.

 Example: `google.com`, `facebook.com`

- It is a **human-friendly name** that points to a **server IP address**.
- The **Domain Name System (DNS)** translates the domain name into an IP address (like `142.250.182.78`).

📝 Instead of typing an IP address, you just type the domain name.

---

4. What is Hosting?

Hosting means **storing website files** (HTML, images, databases) on a **server** that is connected to the Internet.

- A **hosting provider** gives you space on a server so your website can be accessed online.
- Common hosting companies: **Hostinger**, **GoDaddy**, **Bluehost**, **AWS**

🛠 Types of Hosting:

- **Shared Hosting** – Many websites on one server (cheap, basic)
- **VPS Hosting** – Private space on a shared server
- **Dedicated Hosting** – Entire server for one website (expensive)
- **Cloud Hosting** – Scalable and flexible (like Google Cloud, AWS)

Module 2 – Fundamentals of World Wide Web

THEORY EXERCISE:

1. Difference between Web Designer and Web Developer

Web Designer	Web Developer
Focuses on the **look and feel** of a website (design, layout, colors).	Focuses on the **functionality and coding** of the website.
Uses tools like **Adobe XD, Figma, Photoshop**.	Uses programming languages like **HTML, CSS, JavaScript, PHP, Python**.
Creates **UI/UX** (user interface/user experience).	Builds the **structure and logic** behind the website.
Example: Makes sure the website looks good on all screen sizes.	Example: Makes the contact form work and stores the data.

🖌 Designer = Artist
🦾💻 Developer = Engineer

---

2. What is W3C?

W3C stands for **World Wide Web Consortium**.

- It is an **international organization** that develops **web standards**.
- Created by **Tim Berners-Lee** (the inventor of the web).
- Its goal is to make the **web accessible, secure, and usable** for everyone.

💡 Example: W3C defines how HTML, CSS, and other web technologies should work.

---

3. What is a Domain?

A **domain** is the **name** of a website that people type in the address bar.

- It is linked to the website's IP address.
- Example: `www.amazon.com` is a domain.
- Domains are **purchased from registrars** (like GoDaddy or Namecheap).

🔤 **Parts of a domain:**

- **www** – Subdomain
- **example** – Domain name
- **.com** – Extension (TLD)

---

4. What is SEO?

SEO stands for **Search Engine Optimization**.

- It is the process of **improving a website** so it ranks higher on search engines like Google.
- Goal: **Increase visibility**, get more visitors.

🛠 **Types of SEO:**

- **On-page SEO** – Content, keywords, titles, images
- **Off-page SEO** – Backlinks, social sharing
- **Technical SEO** – Website speed, mobile-friendliness, code quality

📈 Good SEO = More traffic = More business

---

5. What is SDLC (Software Development Life Cycle)?

SDLC is the process of **developing software step by step**.

🧿 Phases of SDLC:

1. **Requirement Gathering** – What do users need?
2. **Planning** – Time, cost, resources
3. **Design** – Create architecture, database, UI
4. **Development** – Write code
5. **Testing** – Check for bugs and fix errors
6. **Deployment** – Launch the software
7. **Maintenance** – Update, fix issues after launch

🎹 It helps in building software **efficiently and systematically**.

Module 3 – Fundamentals of IT

THEORY EXERCISE:

📱 What is a Program? (In Simple Words)

A **program** is a set of **instructions** written by a person (a programmer) to make a **computer do a specific task**.

Just like a recipe tells a chef what to do step-by-step, a program tells a computer what actions to perform, like:

- Showing a message on screen
- Adding two numbers
- Opening a website
- Saving a file

⚙ How Does a Program Function?

1. **Written in a Language Computers Understand**
   Programs are written using **programming languages** like Python, C++, or Java.
2. **Stored as Code**
   The code is saved as a file (like `.py`, `.cpp`, etc.).

3. **Run by a Processor**
   When you run the program:
   - The **CPU (central processing unit)** reads each instruction **one by one**.
   - It performs the task: math, showing text, saving data, etc.
4. **Input → Process → Output**
   A program usually follows this basic flow:
   - **Input**: Data from user or file
   - **Process**: Computer thinks/calculates
   - **Output**: Shows result on screen or stores it

✅ Example:

Imagine a calculator app:

- You enter: 5 + 3
- The program reads your input
- It processes the addition
- It shows: **8**

THEORY EXERCISE:

: **What are the key steps involved in the programming process?**

✅ 1. Understanding the Problem

- First, you must clearly understand **what the program should do**.
- Example: "Create a program to calculate the average of three numbers."

✅ 2. Planning the Solution

- Think and plan **how the program will solve the problem**.
- You may use:
  - **Flowcharts** (diagrams)
  - **Pseudocode** (simple English instructions)

✅ 3. Writing the Code

- Use a **programming language** (like Python, C++, Java) to write the instructions.
- This is called **coding or development**.

✅ 4. Compiling or Interpreting

- The code needs to be **converted** into a form the computer understands (machine code).
- Some languages use a **compiler** (like C++).
- Others use an **interpreter** (like Python).

✅ 5. Testing the Program

- Run the program with different inputs to **check for errors (bugs)**.
- Make sure it gives the correct output.

✅ 6. Debugging

- If there are any mistakes or bugs, **find and fix them**.

✅ 7. Final Execution

- Once it works correctly, the program is ready to be **used by others** or **deployed** to a real system.

✅ 8. Maintenance and Updates

- After launch, you may need to:
  - Add new features
  - Fix problems
  - Improve performance

THEORY EXERCISE:

What are the main differences between high-level and low-level programminglanguages?

🏭 1. Level of Abstraction

High-Level Language	Low-Level Language
Closer to **human language**	Closer to **machine language**
Easy to read and write	Harder to read and understand
Example: Python → `print("Hello")`	Example: Assembly → `MOV AX, 4C00h`

🛠 2. Ease of Use

High-Level	Low-Level
Easier to learn for beginners	Requires technical knowledge of hardware
Automatic memory management (in most cases)	Manual memory and CPU management

---

⚡ 3. Speed and Performance

High-Level	Low-Level
Slower execution (due to abstraction)	Very fast and efficient
Compiled or interpreted into machine code	Already close to machine code

---

□ 4. Hardware Control

High-Level	Low-Level
Limited control over hardware	Full control over memory and hardware
Used for apps, websites, games	Used for systems, firmware, drivers

---

□ 5. Examples

- **High-Level Languages**: Python, Java, C++, JavaScript, PHP
- **Low-Level Languages**: Assembly language, Machine code (binary)