Module 1: Basics of HTML

1. Introduction to HTML

Definition: HTML stands for **HyperText Markup Language**.

- **HyperText:** "Hyper" means non-linear. It refers to the way web pages are linked together. You can click a link and jump to any page on the internet.
- **Markup Language:** It is *not* a programming language (like Java or Python). It does not have logic (loops/functions). Instead, it uses **tags** to "mark up" text to tell the browser how to display it.
    - *Analogy:* Think of HTML as the **Skeleton** of a human body. It gives structure. (CSS is the skin/clothes, JS is the muscle/brain).

1.1 HTML Syntax and Elements

- **Element:** An HTML element typically consists of a **start tag**, **content**, and an **end tag**.
    - Syntax: <tagname>Content goes here...</tagname>
- **Empty Elements (Void Elements):** Tags that do not have closing tags or content.
    - Examples:
 (Line Break), <hr> (Horizontal Rule), <img> (Image).
- **Attributes:** Provide additional information about an element. They are always specified in the start tag.
    - Format: name="value"
    - Example: Click me (href is the attribute).

1.2 The Document Structure (The Boilerplate)

Every HTML5 document follows a strict tree structure.

```
1. <!DOCTYPE html>          <!-- Tells browser this is HTML5 -->
2. <html lang="en">         <!-- Root element -->
3.   <head>                 <!-- Brain: Meta-data, Title, CSS links (Not visible on page) -->
4.     <meta charset="UTF-8">
5.     <title>My Page</title>
6.   </head>
7.   <body>                 <!-- Body: Everything visible to the user -->
8.     <h1>Hello World</h1>
```

```
 9.  </body>
10. </html>
11.
```

2. Basic HTML Tags

2.1 Headings and Paragraphs

- **Headings (<h1> to <h6>):**
    - <h1> is the most important (Main Title). <h6> is the least important.
    - *SEO Tip:* Use only **one** <h1> per page for Google Search Engine Optimization.
- **Paragraph (<p>):**
    - Defines a block of text. Browsers automatically add margins before and after a paragraph.

2.2 Lists

Organizing data is crucial. HTML provides three types:

1. **Unordered List (<ul>):** Bullet points.
    - *Tags:* wraps the list, <li> (List Item) defines items.
2. **Ordered List (<ol>):** Numbered list (1, 2, 3 or A, B, C).
    - *Attributes:* type="A", start="5".
3. **Definition List (<dl>):** Dictionary-style list.
    - <dt>: Term (e.g., "HTML").
    - <dd>: Definition (e.g., "Standard markup language...").

2.3 Hyperlinks (<a>)

The Anchor tag connects the web.

- **Syntax:** Link Text
- **Attributes:**
    - href: The destination URL.
    - target="_blank": Opens link in a new tab.
    - title: Tooltip text on hover.

2.4 Media Elements (HTML5 Special)

Before HTML5, we needed Flash player for video. Now, it's native.

1. **Images (<img>):**

```
1. <img src="cat.jpg" alt="A cute cat" width="300">
```

   - **alt (Alternative Text):** Crucial for blind users (Screen Readers) and if the image fails to load.

2. **Audio (<audio>):**

```
1. <audio controls src="music.mp3">Your browser does not support audio.</audio>
```

   - Attributes: controls (play/pause buttons), autoplay, loop.

3. **Video (<video>):**

```
1. <video width="320" controls poster="thumbnail.jpg"> <source src="movie.mp4"
type="video/mp4"> </video>
```

3. Advanced HTML: Tables and Forms

3.1 HTML Tables

Used for structured data (like a calendar or mark sheet), NOT for layout.

- **Structure:**
  - <table>: Container.
  - <tr>: Table Row.
  - <th>: Table Header (Bold and centered by default).
  - <td>: Table Data (Regular cell).
- **Merging Cells:**
  - rowspan="2": Merge two cells vertically.
  - colspan="2": Merge two cells horizontally.

3.2 HTML Forms (Crucial for PHP Module)

Forms allow users to send data to the server.

- **The <form> Container:**
    - action="login.php": Where the data goes.
    - method="GET": Data is sent in URL (insecure, good for search).
    - method="POST": Data is hidden in body (secure, good for passwords).
- **Form Elements:**

1. **Input (<input>):** The swiss-army knife.

```
1. type="text" (Name), type="password" (Dots), type="email" (Validation), type="date",
type="file", type="submit".
2.
```

2. **Textarea (<textarea>):** Multi-line input (e.g., Comments).

3. **Select & Option (Dropdown):**

```
1. <select name="city">
2.     <option value="patna">Patna</option>
3.     <option value="delhi">Delhi</option>
4. </select>
5.
```

4. **Fieldset & Legend:** Groups related elements with a border.

```
1. <fieldset>
2.     <legend>Personal Info</legend>
3.     <input type="text" placeholder="Name">
4. </fieldset>
5.
```

4. Semantic HTML (The Modern Web)

Definition: Semantic elements clearly describe their meaning to both the browser and the developer.

- **Non-Semantic:** and tells nothing about its content.
- **Semantic:** <form>, <table>, <article> tells exactly what's inside.

4.1 Block vs. Inline Elements (Deep Concept)

- **Block-level Elements (, <p>, <h1>, <section>):**
    - Always start on a new line.
    - Take up the full width available (stretch left-to-right).

- **Inline Elements (, <a>, <b>, <img>):**
    - Do not start on a new line.
    - Take up only as much width as necessary.

4.2 Major Semantic Tags (HTML5 Layout)

Instead of using , HTML5 gives us specific tags.

1. **<header>:** Intro content (Logo, Nav). Not just for the top of the page, can be the header of an article too.
2. **<nav>:** Container for navigation links.
3. **<section>:** A thematic grouping of content (e.g., "Services", "Contact Us").
4. **<article>:** Independent, self-contained content (e.g., A Blog Post, A News Item). It should make sense even if taken out of the website.
5. **<aside>:** Content aside from the main content (Sidebar, Ads).
6. **<footer>:** Copyright info, contact links.

5. Navigation Bar Creation

A navbar is essentially a list of links styled to look like a bar.

Step 1: Structure (HTML) We use a <nav> containing an Unordered List (<ul>).

```
<nav>
    <ul>
        <li><a href="#home">Home</a></li>
        <li><a href="#services">Services</a></li>
        <li><a href="#contact">Contact</a></li>
    </ul>
</nav>
```

Module II: Styling with CSS and Bootstrap

1. Introduction to CSS (Cascading Style Sheets)

Definition: CSS is a style sheet language used for describing the presentation of a document written in HTML.

- **HTML:** Content (Skeleton).
- **CSS:** Presentation (Skin/Clothes).
- **Why "Cascading"?** It means rules "cascade" down. If you have conflicting styles (e.g., one rule says "Blue", another says "Red"), a specific priority system determines which one wins.

1.1 Types of CSS (Where to write it)

1. **Inline CSS:** Written inside the HTML tag.

```
1. <h1 style="color: blue;">Hello</h1>
```

   - Pros: Quick fix. Cons: Hard to maintain, no reusability.

2. **Internal CSS:** Written inside the <style> tag in the <head>.

   - Pros: Good for single-page sites. Cons: Cannot be shared across pages.

3. **External CSS (Best Practice):** Written in a separate .css file (e.g., style.css) and linked.

```
1. <link rel="stylesheet" href="style.css">
```

   - Pros: Reusability, cleaner HTML, faster loading (browser caches the file).

1.2 CSS Selectors (Targeting Elements)

How do you tell CSS *which* element to style?

1. **Element Selector:** Targets all tags of a type.

   - p { color: red; } (All paragraphs become red).

2. **Class Selector:** Targets elements with a specific class attribute. Starts with a dot (.).

   - .btn { background: green; } (Any tag with class="btn").
   - Note: Classes can be reused on multiple elements.

3. **ID Selector:** Targets a **unique** element with a specific ID. Starts with a hash (#).

   - #header { height: 100px; } (Only the tag with id="header").
   - Note: IDs must be unique per page.

4. **Descendant Selector:** Targets nested elements.

```
1. div p { color: yellow; } (Selects <p> only if it is inside a <div>).
```

5. **Universal Selector (*):** Selects everything. Used for resetting margins.

```
1. * { margin: 0; padding: 0; }
```

1.3 The Box Model (Crucial Concept)

Every element in web design is a rectangular box.

The 4 Layers:

1. **Content:** The actual text/image.
2. **Padding:** Space between content and border (Inside the box). Transparent.
3. **Border:** A line around the padding and content.
4. **Margin:** Space outside the border (Distance from neighbors). Transparent.

Example:

```
1. .box {
2.     width: 300px;
3.     padding: 20px; /* Content width stays 300px? No! Total width = 300 + 20+20 = 340px */
4.     border: 5px solid black;
5.     margin: 10px;
6. }
```

- **box-sizing: border-box;:** This property forces the padding and border to be *included* in the width. It makes layout math much easier.

2. Layout and Positioning

2.1 Display Property

Controls how an element behaves in the flow.

- **block:** Starts on a new line, takes full width (e.g., , <p>).
- **inline:** Sits on the same line, takes only needed width (e.g., , <a>). Width/Height don't work.
- **inline-block:** Sits on the same line BUT allows setting Width/Height.
- **none:** Hides the element completely (it disappears from the DOM layout).

2.2 Positioning (position)

1. **static (Default):** Normal flow. top/left don't work.

2. **relative:** Positioned relative to its *normal* position. Moves without affecting neighbors.
3. **absolute:** Removed from normal flow. Positioned relative to the nearest *positioned ancestor* (usually a relative parent).
4. **fixed:** Stuck to the viewport (screen). Doesn't move when scrolling (e.g., Sticky Navbar).
5. **z-index:** Controls the stack order (Front/Back). Only works on positioned elements (not static). Higher number = On top.

2.3 Float (Legacy Layout)

- float: left; pushes an element to the left, allowing text to wrap around it.
- Historically used for columns, but now replaced by **Flexbox** and **Grid**.

3. Responsive Web Design (RWD)

Definition: Designing websites that look good on all devices (Desktops, Tablets, Phones).

3.1 The Viewport Meta Tag

You MUST include this in HTML <head> for responsiveness to work: <meta name="viewport" content="width=device-width, initial-scale=1.0">

3.2 Media Queries (@media)

CSS rules that apply only when specific conditions are true.

```
/* Default styles for Mobile */
body { background: white; font-size: 14px; }

/* Styles for Tablets and up (min-width: 768px) */
@media (min-width: 768px) {
    body { background: lightblue; font-size: 16px; }
}

/* Styles for Desktops (min-width: 1024px) */
@media (min-width: 1024px) {
    body { background: lightgreen; font-size: 18px; }
}

```

4. Bootstrap Framework (The Fast Lane)

Definition: Bootstrap is the most popular CSS Framework. It gives you pre-written CSS classes so you don't have to code layouts from scratch.

- **Installation:** Link the Bootstrap CDN (Content Delivery Network) in your <head>.

4.1 The Grid System (12 Columns)

Bootstrap divides the screen width into **12 equal columns**.

- **Rows:** Columns must be inside a .row.
- **Classes:** .col- followed by breakpoint (sm, md, lg) and size (1-12).

Example: 3 Equal Columns on Desktop, Stacked on Mobile

```
<div class="container">
  <div class="row">
    <div class="col-md-4">Column 1</div>
    <div class="col-md-4">Column 2</div>
    <div class="col-md-4">Column 3</div>
  </div>
</div>

```

- On a phone (< 768px), col-md-4 is ignored, and divs stack (100% width).
- On a laptop (>= 768px), each takes 4/12 (33%) width.

4.2 Key Bootstrap Components (Classes)

1. **Buttons:**

```
<button class="btn btn-primary">Blue</button>
<button class="btn btn-danger">Red</button>
```

2. **Forms:**
    - Use .form-control on inputs for full-width, styled fields.
3. **Navbar:**

```
.navbar, .navbar-expand-lg, .navbar-dark, .bg-dark.

```

    - It automatically creates a "Hamburger Menu" on mobile.
4. **Cards:**
    - A flexible content container with image, text, and button.

```
.card, .card-body, .card-title.
```

5. **Modals:**
    - Pop-up dialog boxes that overlay the page.

Teacher's Homework for Module II:

1. **CSS Layout Task:** Create a layout with a Header, Sidebar, Main Content, and Footer using float or flexbox. Use different background colors to visualize the Box Model.
2. **Bootstrap Task:** Re-create your Resume from Module 1, but this time use Bootstrap Grid (Left column for personal info, Right column for experience) and Buttons.

Module III: JavaScript and Client-Side Scripting

1. JavaScript Basics

Definition: JavaScript (JS) is a lightweight, interpreted, object-oriented language with first-class functions. While HTML structures the page and CSS styles it, JavaScript breathes life into it.

- **Client-Side Scripting:** JS runs primarily on the user's browser (Client) using engines like **V8 (Chrome)** or **SpiderMonkey (Firefox)**. It does not require a server to execute code, which reduces server load and provides instant feedback (e.g., validating a password strength instantly).
- **Java vs. JavaScript:** It is crucial to remember they are unrelated.
    - **Java:** Statically typed, compiled, runs on JVM.
    - **JavaScript:** Dynamically typed, interpreted, runs in the browser.

1.1 Variables and Scope (The "Let vs Var" Debate)

In modern JavaScript (ES6+), how you declare variables matters significantly for memory and scope.

- **var (Legacy/Function Scope):**
    - Has **Function Scope**. If declared inside a block if(true) { var x = 10; }, x leaks out and is accessible outside the block.
    - **Hoisting:** Variables declared with var are moved to the top of their scope during compilation. You can use them before declaration (result is undefined).
- **let (Modern/Block Scope):**

- Has **Block Scope**. If declared inside { let y = 10; }, y dies when the block ends. This prevents accidental overwrites.
- Does **not** allow hoisting in the same way (Temporal Dead Zone).

- **const (Constant):**
  - Value cannot be reassigned.
  - *Nuance:* If const holds an Object or Array, the *contents* can change, but the *reference* cannot.
  - *Example:* const arr = [1, 2]; arr.push(3); is Allowed. arr = [4, 5]; is Error.

1.2 Data Types: Primitives vs. References

- **Primitive Types (Stored in Stack):** Immutable data.
  - **String:** "Amity University"
  - **Number:** All numbers are floating point (64-bit) in JS. 10 and 10.5 are the same type.
  - **Boolean:** true, false.
  - **Undefined:** A variable declared but not assigned a value.
  - **Null:** An assignment value that represents "no value" or "empty object".
- **Reference Types (Stored in Heap):** Mutable data.
  - **Object:** { id: 101, name: "Rahul" }.
  - **Array:** An ordered list of values. [10, 20, "Text"] (Arrays can hold mixed types).

1.3 Operators and Type Coercion

JavaScript tries to be helpful by converting types automatically, which leads to confusing bugs (Type Coercion).

- **Equality Operators:**
  - **== (Loose Equality):** Converts types before comparing.

```
1. 5 == "5" =>true (String "5" becomes Number 5).
2. 0 == false =>true.
```

  - **=== (Strict Equality):** Checks Value AND Type. **Always use this.**

```
1. 5 === "5" =>false.
```

- **Ternary Operator:** A shorthand for if-else.
    - Syntax: condition ? valueIfTrue : valueIfFalse
    - Example: let status = (age >= 18) ? "Adult" : "Minor";

1.4 Control Structures and Functions

- **Switch Statement:** Useful for multiple conditions.

```
1. switch(day) {
2.     case 0: text = "Sunday"; break;
3.     case 6: text = "Saturday"; break;
4.     default: text = "Weekday";
5. }
6.
```

- **Functions (3 Ways to Write):**
    1. **Function Declaration:** function add(a, b) { return a + b; } (Hoisted).
    2. **Function Expression:** const add = function(a, b) { return a + b; }; (Not Hoisted).
    3. **Arrow Functions (ES6):** Concise syntax. const add = (a, b) => a + b;

2. The DOM (Document Object Model)

Definition: The DOM is an Application Programming Interface (API) for HTML. When a web page is loaded, the browser creates a **Document Object Model** of the page. It is constructed as a tree of Objects.

2.1 Advanced Selection Methods

- **document.getElementById('id'):** Returns a single object. Fast.
- **document.getElementsByClassName('class'):** Returns an HTMLCollection (Array-like list). You must use an index [0] to access items.
- **document.querySelectorAll('.css-selector'):** Returns a NodeList of all matches.
    - Example: document.querySelectorAll('div > p') selects all paragraphs directly inside divs.

2.2 DOM Traversal and Manipulation

We can move through the tree and change it dynamically.

- **Traversal:**
    - element.parentNode: Selects the parent.

- element.children: Selects child elements.
- element.nextElementSibling: Selects the next tag.

- **Creation & Insertion (Dynamic Content):**

```
// 1. Create a new element
let newBtn = document.createElement("button");

// 2. Add content
newBtn.innerHTML = "Click Me";

// 3. Append to Body
document.body.appendChild(newBtn);

// 4. Remove Element
document.body.removeChild(newBtn);

```

3. Events and Event Handling

Definition: Interaction with the web page triggers events. The browser "fires" an event, and we "listen" for it.

3.1 The Event Object (e)

When an event occurs, JS automatically passes an **Event Object** to the function. This object contains details about the event.

- e.target: The specific HTML element that was clicked.
- e.type: The type of event (e.g., "click").
- e.preventDefault(): Stops the default browser behavior (e.g., stops a form from submitting or a link from opening).

3.2 Event Propagation: Bubbling vs. Capturing

This is a deep concept often asked in exams. When you click a <button> inside a , which one gets the click first?

1. **Bubbling (Default):** The innermost element handles the event first, then it "bubbles up" to parents.
    - Button Clicked =>Div Clicked =>Body Clicked.
2. **Capturing:** The outermost element handles it first, then "trickles down."
    - Body =>Div =>Button.

3.3 Handling Events (The addEventListener Way)

Why use addEventListener instead of onclick=""?

1. **Multiple Listeners:** You can attach 5 different functions to the same "click" event.
2. **Separation of Concerns:** Keeps HTML clean (Structure) and JS separate (Behavior).

```
let btn = document.getElementById("myBtn");
btn.addEventListener("click", function(e) {
    console.log("Button Clicked at coordinates: " + e.clientX + "," + e.clientY);
});

```

4. Output Methods and Security

- **window.alert():** Blocks code execution until the user clicks OK. Poor UX.
- **document.write(): Dangerous.** It overwrites the entire HTML document if called after the page has loaded. Only use for testing.
- **innerHTML:** The standard way to update content.
    - Security Warning: Be careful of **XSS (Cross-Site Scripting)**. If you insert user input directly into innerHTML, a hacker can insert a <script> tag to steal cookies. Use textContent for plain text.

5. Form Validation (Client-Side)

Definition: Validating input at the client side provides a better User Experience (UX) by catching errors instantly without a server round-trip.

5.1 Regular Expressions (Regex)

For complex validation (like Emails or Passwords), simple logic isn't enough. We use Regex patterns.

- **Email Pattern:** /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/
- **Method:** pattern.test(string) returns true or false.

Example: Advanced Login Validation

```
<form id="loginForm" onsubmit="return validate()">
    Username: <input type="text" id="user"><br>
    Email: <input type="text" id="email"><br>
    <span id="errorMsg" style="color:red"></span><br>
    <input type="submit">
</form>

<script>
function validate() {
```

```
    let u = document.getElementById("user").value;
    let e = document.getElementById("email").value;
    let msg = document.getElementById("errorMsg");

    // 1. Empty Check
    if (u === "" || e === "") {
        msg.innerHTML = "All fields are mandatory!";
        return false; // Stop Submission
    }

    // 2. Regex Email Check
    let emailPattern = /^\S+@\S+\.\S+$/; // Simple check for @ and .
    if (!emailPattern.test(e)) {
        msg.innerHTML = "Invalid Email Format!";
        return false;
    }

    alert("Form is valid! Submitting...");
    return true;
}
</script>

```

Module IV: Advanced Web Technologies - XML, AJAX, and JSON

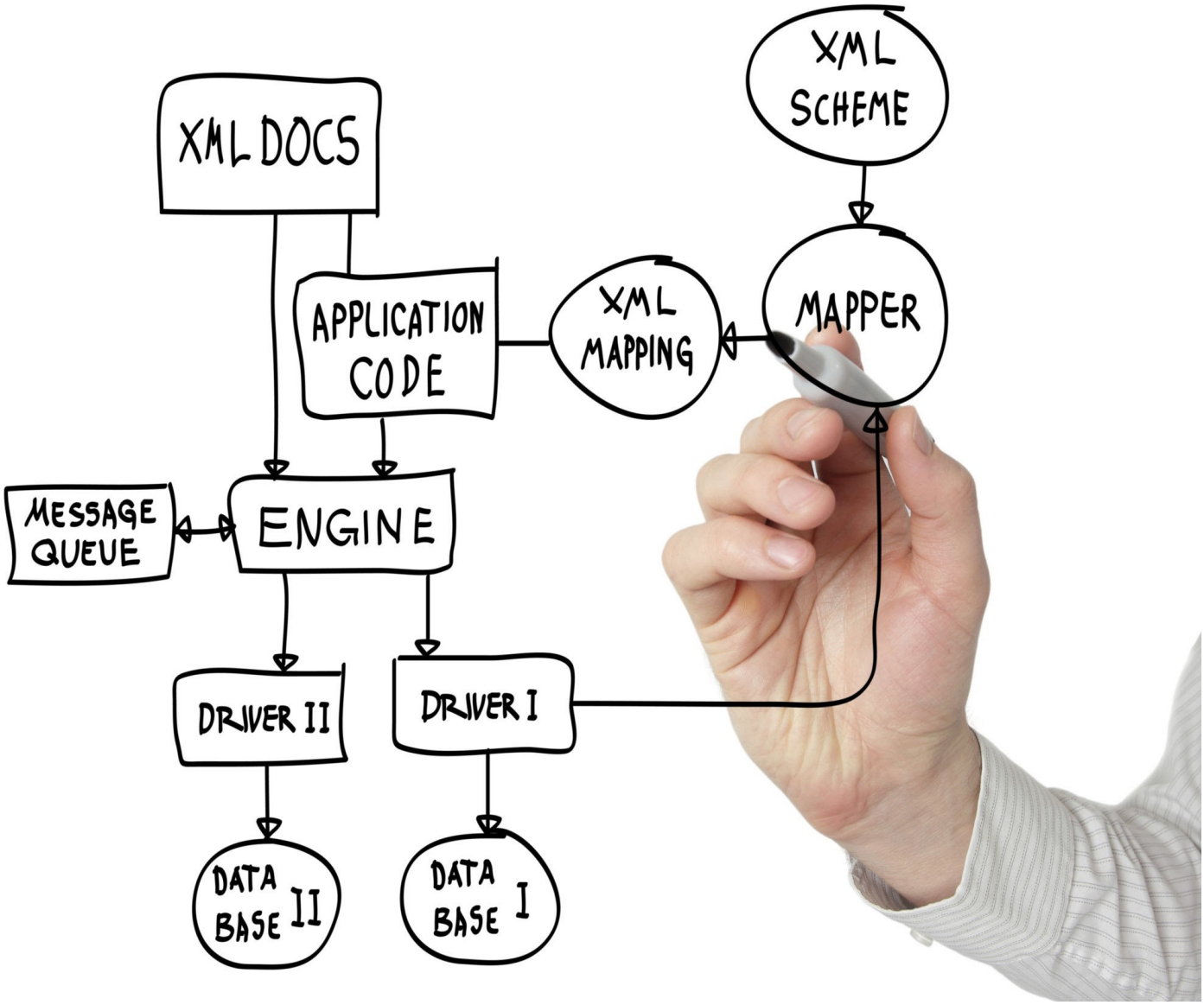
1. XML (eXtensible Markup Language)

Definition: *XML is a markup language designed to store, transport, and structure data. Unlike HTML, which dictates how data looks (bold, italic, etc.), XML dictates what the data is.*

- ***Design Goal:*** *XML is software- and hardware-independent. It acts as a universal translator, allowing two incompatible systems (e.g., a Java backend and a Python script) to exchange data seamlessly.*
- ***Self-Descriptive:*** *XML tags are not predefined. You must "invent" your own tags that describe the data they hold.*
    - *Example: <student><name>Rahul</name></student> tells you exactly what the data represents. In HTML, you might just see Rahul , which conveys no semantic meaning.*

1.1 XML Structure & Syntax

An XML document forms a strict ***Tree Structure*** *that starts at "the root" and branches to "the leaves". This hierarchy makes it easy for programs to traverse the data.*



Crucial Syntax Rules (Strictness):

1. ***Root Element:*** *Every XML document must have exactly one root element that wraps everything else.*
2. ***Closing Tags:*** *All tags must be closed. <tag>Data</tag>. Omitting a closing tag breaks the file.*
3. ***Case-Sensitivity:*** *<Tag> is different from <tag>. A mismatch causes a parsing error.*
4. ***Nesting:*** *Elements must be properly nested. Text is wrong; Text is correct.*
5. ***Attribute Quoting:*** *Attribute values must always be in quotes. <note date="12-01-2023"> is correct; <note date=12-01-2023> is incorrect.*

Example: A University Data Tree

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Root Element -->
<university>
  <!-- Child Element -->
  <department name="CSIT">
    <!-- Sibling Elements -->
    <student id="101">
      <name>Amit</name>
      <cgpa>8.5</cgpa>
      <contact>
         <email>amit@amity.edu</email>
      </contact>
    </student>
    <student id="102">
      <name>Priya</name>
      <cgpa>9.1</cgpa>
    </student>
  </department>
</university>

```

1·2 Attributes vs· Elements (Design Choice)

In the example above, id="101" is an attribute, while <name> is an element·

- ***Guideline:*** *Use* ***attributes*** *for metadata (data about data, like IDs)· Use* ***elements*** *for the data itself· Elements are more flexible because they can be extended (e·g·, <name> can be split into <first> and <last>)·*

1·3 DTD (Document Type Definition) & Schema

Since XML allows custom tags, how do we ensure everyone uses the same structure? We use a blueprint to enforce rules·

- ***Well-Formed vs· Valid XML:***
  - ***Well-Formed:*** *Follows the basic syntax rules (tags closed, nested properly)·*
  - ***Valid:*** *Follows a specific DTD or Schema·*
- ***DTD:*** *The older way to define structure·*
  - *Example Rule: <!ELEMENT student (name, cgpa, contact?)> implies a student must have a name and cgpa, but contact is optional (?)·*
- ***XML Schema (XSD):*** *The modern, XML-based alternative· It allows for data typing (e·g·, defining that <cgpa> must be a Decimal between 0 and 10)·*

1·4 XML Parsing & Transformation

- **Parsing:** The process of reading an XML document so code can use it.
    - **DOM Parser:** Loads the *entire* XML tree into memory. Good for editing, bad for massive files.
    - **SAX Parser:** Reads the file line-by-line. Good for reading massive files, but cannot edit.
- **XSLT (eXtensible Stylesheet Language Transformations):**
    - XML stores data. XSLT transforms that data into HTML for display or into another XML format.
    - *Analogy:* XML is the raw ingredients (flour, sugar); XSLT is the recipe that cooks it into a meal (Web Page).

2. XHTML (eXtensible HyperText Markup Language)

Definition: XHTML is HTML written as XML. It acts as a bridge between the sloppy syntax of old HTML and the strict syntax of XML.

- **Why was it needed?** Early HTML browsers were too forgiving (they would display a page even if you forgot </html>). This made parsing difficult for small devices like early mobile phones. XHTML forced strict syntax to ensure consistency across all platforms.
- **Key Differences from HTML:**
    1. **DOCTYPE:** <!DOCTYPE> is mandatory.
    2. **Structure:** <html>, <head>, <body> are mandatory.
    3. **Self-Closing:** Elements like
 and <img> must be self-closed (
).
    4. **Case:** Element and attribute names must be in lower case.

3. AJAX (Asynchronous JavaScript And XML)

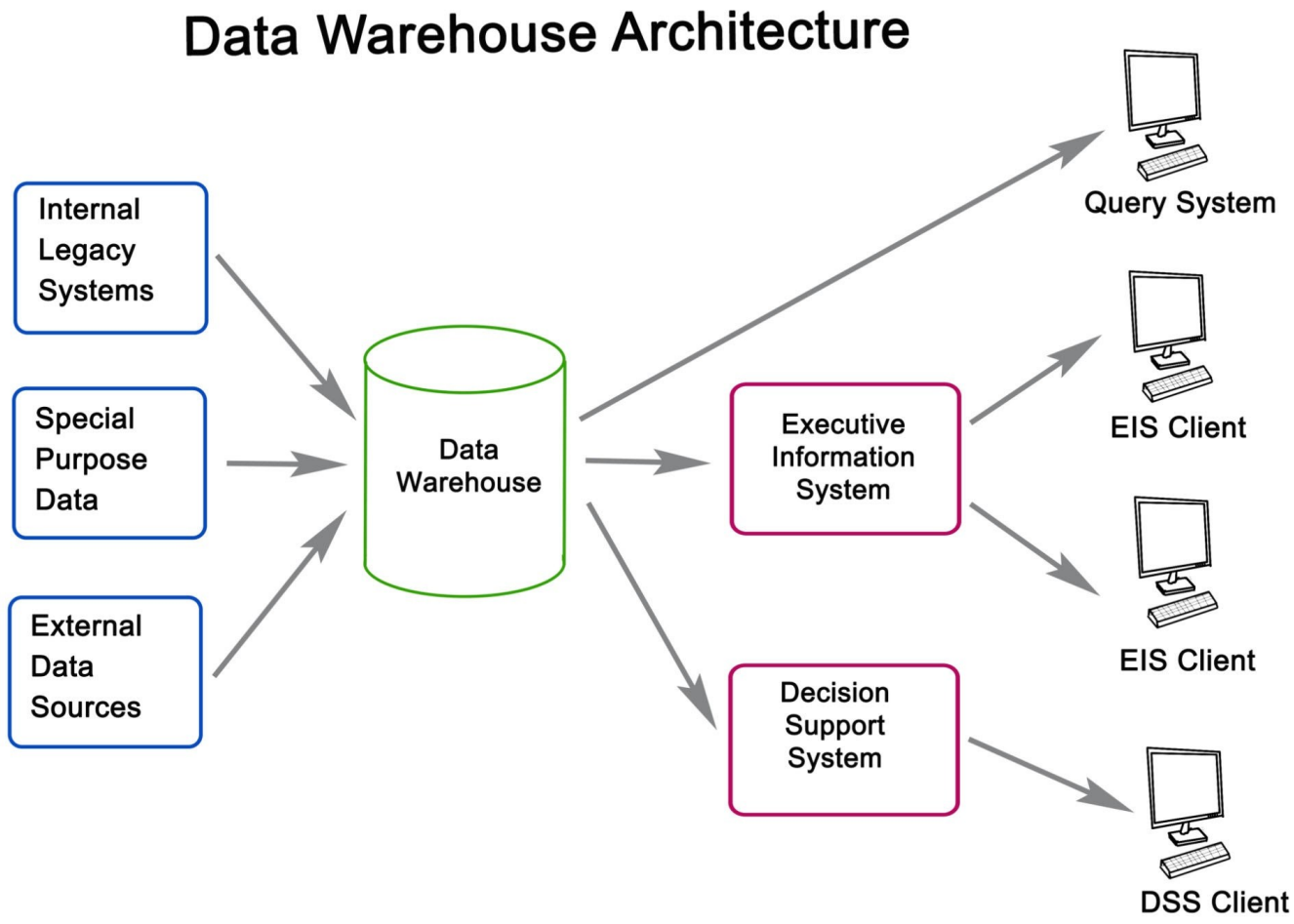
Definition: AJAX is not a single programming language. It is a technique combining XMLHttpRequest (for data transport) and JavaScript (for logic) to create fast, dynamic web interfaces.

- **The "Asynchronous" Magic:** In a traditional web model, if you click a button, the entire page reloads (screen flashes white). In AJAX, the browser talks to the server in the background (Asynchronously) while the user continues to interact with the page. This prevents "blocking" the user experience.
- **Real-World Examples:**

- **Google Search:** Autosuggestions appear as you type without reloading.
- **Instagram:** Infinite scroll loads more photos as you reach the bottom.

3.1 How AJAX Works (The Workflow)

1. **Event:** An event occurs (e.g., button click).
2. **Request:** JS creates an XMLHttpRequest object.
3. **Send:** The request is sent to the server (e.g., "Give me data.txt").
4. **Process:** The server finds the file/data.
5. **Response:** The server sends data back.
6. **Update:** JS accepts the data and uses DOM Manipulation (innerHTML) to update just a specific .



3.2 The XMLHttpRequest Object Properties

Understanding the internal state of this object is crucial for debugging.

- **readyState:** Tracks the status of the request.
    - 0: UNSENT (Request not initialized).
    - 1: OPENED (Connection established).
    - 2: HEADERS_RECEIVED.
    - 3: LOADING (Downloading data).
    - 4: DONE (Operation complete).
- **status:** HTTP Status code.
    - 200: OK (Success).
    - 404: Not Found.
    - 500: Server Error.

Example Code:

```
1. function loadDoc() {
2.   const xhttp = new XMLHttpRequest();
3.
4.   // Function calls every time readyState changes
5.   xhttp.onreadystatechange = function() {
6.     // Check if download is DONE (4) and Successful (200)
7.     if (this.readyState == 4 && this.status == 200) {
8.        document.getElementById("demo").innerHTML = this.responseText;
9.     }
10.  };
11.
12.  xhttp.open("GET", "ajax_info.txt", true); // true = Asynchronous
13.  xhttp.send(); // Send request
14. }
15.
```

4. JSON (JavaScript Object Notation)

Definition: JSON is a lightweight data-interchange format derived from JavaScript object syntax. However, it is **language-independent** (Python, Java, and C++ all have libraries to read JSON).

- **The XML Killer:** JSON has largely replaced XML for Web APIs (REST) because:
    1. It is less verbose (no closing tags).
    2. It maps directly to data structures (Arrays/Objects) used in coding.

4.1 JSON Syntax Rules

1. Data is in **key/value** pairs ("key": value).
2. Data is separated by **commas**.
3. Curly braces {} hold **objects**.
4. Square brackets [] hold **arrays**.
5. **Strictness:** Keys must be strings wrapped in **double quotes**. Single quotes are not allowed in JSON.

Example: Complex JSON Data

```
{
  "university": "Amity Patna",
  "established": 2017,
  "departments": [
    { "name": "CSIT", "head": "Dr. Sharma" },
    { "name": "Management", "head": "Dr. Singh" }
  ],
  "isActive": true
}

```

4.2 Converting Data (Serialization vs Deserialization)

In web apps, data travels as text (String). We need to convert between "String" and "Object".

1. **JSON.parse() (Deserialization):** Converts a JSON string (from server) into a JavaScript Object.
2. const jsonString = '{"name": "Rahul", "age": 21}';
3. const obj = JSON.parse(jsonString);
4. console.log(obj.name); // Output: Rahul
5. **JSON.stringify() (Serialization):** Converts a JavaScript Object into a JSON string (to send to server).
6. const obj = { name: "Rahul", age: 21 };
7. const myJSON = JSON.stringify(obj);
8. // Output: '{"name":"Rahul","age":21}'

4.3 JSON vs. XML (Exam Comparison)

Feature	JSON	XML
Size	Lightweight (Compact).	Heavy (Tags add character count).
Parsing Speed	Fast (Native JS engine).	Slow (XML DOM requires traversal).
Readability	Clean, code-like.	Verbose, document-like.
Data Types	String, Number, Array, Boolean, Null.	Everything is treated as a Text Node.
Comments	Not supported.	Supported <!-- comment -->.
Usage	REST APIs, Config files.	SOAP APIs, Document storage.

4.4 Fetching JSON Data (Modern API)

The fetch() API is the modern replacement for XMLHttpRequest. It uses **Promises** to handle asynchronous results cleaner.

```
1. fetch('[https://api.example.com/data.json](https://api.example.com/data.json)')
2.   .then(response => response.json()) // Convert response stream to JSON
3.   .then(data => {
4.       console.log(data.name); // Access data directly
5.   })
6.   .catch(error => console.error('Error:', error)); // Handle errors
```

Module V: PHP Programming & Working With Database

1. Introduction to PHP (Hypertext Preprocessor)

Definition: PHP is a widely-used, open-source, **server-side** scripting language designed primarily for web development. Unlike client-side languages (like JavaScript) that run in the browser, PHP code is executed on the server, generating HTML which is then sent to the client.

- **The "Server-Side" Logic:**
    - When a user requests a file like profile.php, the web server (Apache/Nginx) does not send the file directly.
    - It passes the file to the **PHP Processor**.
    - The PHP Processor executes the code (e.g., fetches your name from the database).

- It outputs pure HTML (e.g., <h1>Welcome, Rahul</h1>).
- The browser receives only the HTML. The user never sees the source PHP code, making it secure for handling passwords and logic.

- **The LAMP Stack:** PHP is the "P" in the famous LAMP stack (Linux OS, Apache Server, MySQL Database, PHP), powering over 70% of the web (including Facebook and WordPress).

1.2 Basic Syntax & Embedding

PHP can be embedded anywhere inside HTML code.

- **Tags:** Code block starts with <?php and ends with ?>.
- **Statements:** Every command must end with a semicolon ;. Missing this is the most common syntax error.
- **Output Directives:**
  - echo: Faster, can take multiple parameters (echo "A", "B";), has no return value.
  - print: Slower, takes one argument, returns 1 (can be used in expressions).

```
<!DOCTYPE html>
<html>
<body>
    <h1>My First PHP Page</h1>
    <?php
        // This comment is not visible in "View Source" of the browser
        echo "Hello World! ";

        # Variables are case-sensitive ($color is different from $Color)
        $name = "Amity University";
        echo "Welcome to " . $name; // The dot (.) is the concatenation operator
    ?>
</body>
</html>
```

2. Core PHP Concepts

2.1 Variables, Scope, and Constants

- **Dynamic Typing:** PHP is a loosely typed language. You do not declare types (e.g., int $x). The variable type is determined by the value assigned to it.
- **Variable Scope (Context):**
  - **Local Scope:** A variable declared inside a function is *only* visible there.

- **Global Scope:** Declared outside functions. Note: You cannot access a global variable inside a function unless you use the global keyword.
- **Static Scope:** Normally, local variables are deleted when a function ends. A static variable retains its value between function calls.

- **Constants:** Values that cannot change. Defined using define("NAME", value);. Global by default.

2.2 Arrays in PHP (Deep Dive)

PHP arrays are sophisticated map structures. They are far more powerful than C/Java arrays.

1. **Indexed Arrays:** Numeric keys (0, 1, 2...).
2. $colors = ["Red", "Green", "Blue"]; // Short syntax
3. echo count($colors); // Output: 3
4. **Associative Arrays:** Key-Value pairs. Essential for database results.
5. $student = array("name" => "Rahul", "age" => 21, "course" => "B.Tech");
6. echo $student["name"]; // Output: Rahul
7. **Multidimensional Arrays:** Arrays inside arrays.

```
$matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];
```

8. echo $matrix[1][2]; // Output: 6 (Row 1, Col 2)

Essential Array Functions:

- array_push($arr, $val): Adds elements to the end.
- array_merge($arr1, $arr2): Combines arrays.
- sort($arr) / ksort($arr): Sorts by value or key.
- in_array("Val", $arr): Checks existence.

2.3 Superglobals (Global Context Variables)

These are built-in variables always available in all scopes.

- **$_POST vs $_GET:**

- $_GET: Data sent via URL parameters (test.php?id=10). Visible to everyone. Limited size (2048 chars). Used for Search.
- $_POST: Data sent in the HTTP body. Invisible. Unlimited size. Used for Forms/Passwords.

- **$_SESSION:** Stores user information across multiple pages (e.g., "Is User Logged In?"). Stored on the Server.
- **$_COOKIE:** Small data stored on the User's Browser (e.g., "Remember Me").

3. File Handling & Security

PHP allows you to interact with the server's file system. This is powerful but dangerous.

3.1 Modes of Operation

- r: Read only. Pointer at start.
- w: Write only. **Erases** file content. Creates new file if missing.
- a: Append. Write at the end. Preserves content.
- x: Create new file for write. Returns false if file exists.

Reading a File Line-by-Line:

```
1. $file = fopen("data.txt", "r") or die("Unable to open file!");
2. while(!feof($file)) { // Loop until End Of File (feof)
3.   echo fgets($file) . "<br>"; // fgets() reads a single line
4. }
5. fclose($file);
6.
```

3.2 Secure File Uploading (Exam Favorite)

Allowing users to upload files is the biggest security risk (e.g., uploading a .php script instead of an image to hack the site).

Security Checklist:

1. Check $_FILES["file"]["size"] (Limit file size).
2. Check Extension (Allow only JPG, PNG, PDF).
3. Check MIME Type (Verify it's actually an image).
4. Rename the file before saving to prevent overwriting system files.

Secure Upload Script:

```php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["userFile"]["name"]);
$imageFileType = strtolower(pathinfo($target_file, PATHINFO_EXTENSION));

// 1. Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["userFile"]["tmp_name"]);
    if($check === false) { die("File is not an image."); }
}

// 2. Check file size (Limit to 500KB)
if ($_FILES["userFile"]["size"] > 500000) { die("Sorry, your file is too large."); }

// 3. Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png") {
    die("Sorry, only JPG & PNG files are allowed.");
}

// 4. Upload
if (move_uploaded_file($_FILES["userFile"]["tmp_name"], $target_file)) {
    echo "The file ". basename($_FILES["userFile"]["name"]). " has been uploaded.";
}
```

4. Database Integration with MySQL

Definition: *MySQL is the RDBMS (Relational Database Management System). To talk to it, PHP offers two APIs:* ***MySQLi*** *(Improved) and* ***PDO*** *(PHP Data Objects). We focus on MySQLi (Object-Oriented Interface).*

4.1 The Connection Object

We treat the database connection as an object.

```php
$servername = "localhost";
$username = "root";
$password = ""; // Default XAMPP password is empty
$dbname = "collegeDB";

// Create connection using OOP
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
```

4.2 Prepared Statements (Preventing SQL Injection)

Critical Concept: Never insert user input directly into a query like "INSERT... VALUES ('$name')". A hacker can input ' OR 1=1 -- to delete your database (**SQL Injection**).

The Solution: Use Prepared Statements. The SQL logic is compiled first, and data is inserted later

A. Secure Insert (Create)

```php
// 1. Prepare the template (Use ? as placeholders)
$stmt = $conn->prepare("INSERT INTO Students (firstname, email) VALUES (?, ?)");

// 2. Bind parameters (s = string, i = integer, d = double)
$stmt->bind_param("ss", $fname, $email);

// 3. Set values and Execute
$fname = "Rahul";
$email = "rahul@example.com";
$stmt->execute();

$fname = "Priya";
$email = "priya@example.com";
$stmt->execute(); // Reusing the statement is efficient

echo "New records created successfully";
$stmt->close();
```

B. Select Data (Read) Fetching data into an associative array makes it easy to use in HTML.

```php
$sql = "SELECT id, firstname, email FROM Students";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  echo "<table border='1'><tr><th>ID</th><th>Name</th></tr>";
  // Output data of each row
  while($row = $result->fetch_assoc()) {
    echo "<tr><td>" . $row["id"]. "</td><td>" . $row["firstname"]. "</td></tr>";
  }
  echo "</table>";
} else {
  echo "0 results";
}
```

4.3 Managing Sessions (Login System Logic)

Sessions allow you to identify a user as they navigate from page to page.

Logic Flow:

1. **Login Page:** User enters pass. If DB match $\rightarrow$ $_SESSION["user"] = "Rahul";.

2. **Dashboard Page:** Check if (!isset($_SESSION["user"])) { header("Location: login.php"); }.

3. **Logout:** session_destroy();.

Example: dashboard.php

```php
<?php
session_start(); // Must be the VERY first line of code
if(isset($_SESSION["username"])) {
    echo "Welcome " . $_SESSION["username"];
} else {
    echo "Please log in first.";
}
?>
```