

Chapter 1

- **Internet Overview:** The Internet is a global system connecting billions of devices through a network of networks, using the Internet Protocol Suite and technologies like electronic, wireless, and optical networking to link private, public, academic, business, and government systems.
- **Core Function:** It facilitates the exchange of information resources and services, with the World Wide Web (WWW) being its most prominent feature.
- **Historical Roots:** Originated in the 1960s-70s as ARPANET, a US Department of Defense project introducing packet switching (sharing communication paths) and TCP (Transmission Control Protocol) for reliable data transfer.
- **Evolution:** Initially offered electronic mail and file transfers; expanded to universities and commercial use in the late 1980s.
- **World Wide Web Birth:** Invented by Tim Berners-Lee at CERN between 1989-1991, the WWW uses hyperlinked text documents accessed via Hypertext Transfer Protocol (HTTP).
- **Web Structure:** Comprises web pages (text, images, hyperlinks) grouped into websites with common themes or focuses.
- **Browser Milestones:** Early web browsers include Mosaic (1992), Netscape (1994), and Internet Explorer (1995); commercial growth marked by Amazon (1995) and Google (1996).
- **Web Mechanics:** Relies on web servers (e.g., Apache, Microsoft IIS) to deliver pages and web browsers to render them for users.
- **Key Technologies:** Includes URLs (Uniform Resource Locators) for identifying resources, HTTP commands (GET, POST, PUT, DELETE) for server-browser communication, and MIME types (e.g., text/html, image/jpeg) for specifying data formats.
- **Development Tools:** HTML structures content, CSS adds styling, JavaScript enables interactivity on the client side, and PHP powers server-side scripting for dynamic content.
- **Multi-Tier Architecture:** Web applications often split into tiers—client tier (user interface), business logic tier (processing), and data tier (storage, typically in relational databases)—to manage requests and responses efficiently.

Chapter 2: Introduction to HTML

- **Definition and Purpose:**
 - HTML stands for Hypertext Markup Language, the standard markup language for creating web pages.
 - It defines the structure of a website semantically, providing cues for presentation, distinguishing it from programming languages.
 - Web browsers interpret HTML files to render them as visible or audible web pages.
- **Structure of HTML Documents:**
 - Comprises a tree of elements and text, where each element is marked by tags.
 - Tags include a start tag (e.g., `<body>`) and an end tag (e.g., `</body>`), enclosing content.
- **Anatomy of an HTML Document:**
 - `<!DOCTYPE html>`: Declares the document as HTML5, ensuring proper rendering.

- `<html lang="en">`: The root element, with a `lang` attribute specifying the document's language (e.g., English).
- `<head>`: Contains metadata not displayed on the page:
 - `<meta charset="utf-8">`: Defines the character encoding for text rendering.
 - `<title>Hello World</title>`: Sets the browser tab title.
 - `<link href="styles.css" rel="stylesheet">`: Links an external CSS stylesheet.
 - `<script src="jscript.js"></script>`: Links an external JavaScript file.
- `<body>`: Holds visible content, such as:
 - `<h1>Hello World</h1>`: Large heading.
 - `<p>This is a web page.</p>`: Paragraph text.

- **Elements and Tags:**

- **Elements:** Define structure and content (e.g., `<a>` for links, `<p>` for paragraphs, `<div>` for divisions).
- Enclosed in angle brackets (`<` and `>`), with content between opening and closing tags (e.g., `<p>content</p>`).
- **Tags:** Pairs of opening (e.g., `<p>`) and closing (e.g., `</p>`) markers; must be nested correctly without overlap.
- Metadata goes in `<head>`, while content resides in `<body>`.

- **Attributes and Values:**

- **Attributes:** Provide additional information about elements (e.g., `id`, `class`, `src`, `href`).
- Defined within opening tags (e.g., `<a href="demo.html">simple</a>`).
- Format: `attribute="value"` (e.g., `<input name="email" value="who@where.com">`).
- **Void Elements:** Don't require closing tags (e.g., `<input>`, `<hr>`, `<br>`).

- **Classification of HTML Elements:**

- **Top-Level Elements:** `<html>`, `<head>`, `<body>` form the document's core structure.
- **Head Elements:** Non-displayed metadata like `<title>`, `<style>`, `<link>`, `<meta>`, `<base>`, `<script>`.
- **Block-Level Elements:** Occupy 100% width, stack vertically (e.g., `<article>`, `<h1>` to `<h6>`, `<header>`, `<footer>`, `<section>`, `<p>`, `<div>`, `<ul>`, `<table>`, `<form>`).
- **Inline Elements:** Flow horizontally within blocks (e.g., `<a>`, `<audio>`, `<br>`, `<img>`, `<span>`, `<strong>`).

- **Syntax Rules:**

- Tags start with `<` and end with `>`, followed by tag name (e.g., `<p>`).
- Attributes follow the tag name (e.g., `<tag attribute1="value">`); unrecognized tags/attributes are ignored.
- Quotes can use `"` or `'`, but unclosed quotes may blank the page.
- Lowercase markup is best practice; values were historically case-sensitive.
- Void elements can include a trailing `/` (e.g., `<img />`), optional in HTML5 but required for polyglot documents.
- Elements must be well-formed, avoiding bad nesting (e.g., `<p>Stuff <strong></p></strong>` is invalid).

- Attributes can appear in any order, with whitespace allowed around = or between name and attributes.

- **Standard HTML Tags:**

- **Headings:** `<h1>` to `<h6>` for hierarchical titles.
- **Tables:** `<table>`, `<tr>` (row), `<th>` (header), `<td>` (data), `<tbody>`, `<thead>`.
- **Paragraphs:** `<p>` for text blocks.
- **Lists:** `<ol>` (ordered), `<ul>` (unordered), `<li>` (list item).
- **Images:** `<img>` for embedding visuals.
- **Lines:** `<hr>` (horizontal rule), `<br>` (line break).
- **Forms:** `<form>`, `<input>`, `<select>`, `<textarea>` for user input.
- **Links:** `<a>` for hyperlinks.
- **Containers:** `<div>` for grouping content.
- **Formatting:** `<strong>` (bold), `<em>` (italic), `<sub>` (subscript).

- **HTML5 Introduction:**

- Result of collaboration between W3C (World Wide Web Consortium) and WHATWG (Web Hypertext Application Technology Working Group) starting in 2006.
- WHATWG focused on web forms/apps, W3C on XHTML 2.0, merging efforts for a new HTML version.

- **HTML5 Ground Rules:**

- Build on HTML, CSS, DOM, and JavaScript foundations.
- Reduce reliance on external plugins (e.g., Flash).
- Enhance error handling and replace scripting with markup where possible.
- Ensure device independence and transparent development process.

- **HTML5 New Features:**

- **Semantic Elements:** `<header>`, `<footer>`, `<section>`, `<article>` for better structure.
- **Canvas:** `<canvas>` for drawing paths, rectangles, arcs, images, with mouse event support.
- **Media:** `<audio>` and `<video>` elements, including video drawing on canvas.
- **Storage:** Improved local offline storage (beyond cookies).
- **Form Controls:** New input types like `calendar`, `date`, `time`, `email`, `url`, `search`, `color`, etc.
- **APIs:** Geolocation, Drag and Drop, Local Storage, Application Cache, Web Workers, Server-Sent Events (SSE).

- **HTML5 Attributes:**

- Applicable to all elements: `id` (unique identifier), `style` (inline CSS), `class` (style grouping), `title` (tooltip), `hidden` (hides element), `contenteditable`, `draggable`, `dropzone`, `spellcheck`.

- **Semantic Elements in HTML5:**

- Eight new block-level elements: `<header>`, `<section>`, `<footer>`, `<aside>`, `<nav>`, `<main>`, `<article>`, `<figure>`.
- For older browser compatibility, set `display: block` in CSS (e.g., `header { display: block; }`).

- **Migration from HTML4:**

- HTML4 used generic `<div>` with IDs (e.g., `<div id="header">`).
 - HTML5 replaces these with semantic tags (e.g., `<header>`, `<article>`), improving readability and meaning.
-

Chapter 3: Introduction to CSS

- **Definition and Role:**

- Cascading Style Sheets (CSS) is a mechanism for styling web documents, controlling aspects like fonts, colors, and layouts.
- Offers powerful presentation control, enhancing visual appeal and user experience.

- **CSS Structure:**

- Consists of rules, each with one or more **selectors** (targeting elements) and a **declaration block** in curly braces `{ }`.
- Declaration block contains **declarations**: property, colon `:`, value, and semicolon `;`.
- Example: `h1 { font-family: Georgia; font-size: 32px; color: #3099D3; text-align: center; }`.

- **How CSS Works:**

- Selectors identify HTML elements to style, while declarations define the visual properties applied.

- **Methods of Applying CSS:**

- **Inline:** Styles in HTML tags (e.g., `<h3 style="color:blue">Heading</h3>`), least reusable.
- **Embedded:** Within `<style>` tags in `<head>` (e.g., `<style> h3.h3_blue { color: blue; } </style>`), page-specific.
- **External:** Linked via `<link rel="stylesheet" type="text/css" href="my_css.css">`, preferred for consistency across multiple pages.

- **Basic Selectors:**

- **Element:** Targets tags (e.g., `p`, `div`, `h1`).
- **Class:** Uses `.className` (e.g., `.h3_blue`).
- **ID:** Uses `#idName` (e.g., `#my_para`).

- **Advanced Selectors:**

- Universal: `*` (all elements).
- Specific Class/ID: `h1.className`, `h1#idName`.
- Descendant: `p span` (span inside p).
- Child: `p > span` (direct child span of p).
- Group: `h1, h2, p` (multiple elements).
- Attribute: `span[my_id]`, `span[my_id="mspan"]` (elements with specific attributes).

- **Precedence Rules:**

- Priority: Inline > Embedded (Internal) > External.
- Selector hierarchy: ID (highest) > Class > Element.
- **Core CSS Layout Concepts:**
 - Three pillars: **Box Model**, **Floating**, **Positioning**, controlling element arrangement and display.
- **CSS Box Model:**
 - Every element is a box with layers: **Content** (innermost), **Padding** (inside border), **Border**, **Margin** (outside border).
 - **Margins:** Space outside border, can be negative (e.g., `margin: -5px`), affects background positioning.
 - **Padding:** Space inside border, no negative values, shifts content without affecting background.
 - **Shorthand:**
 - Four values: `margin: 0 5px 6px 5px` (top, right, bottom, left).
 - One value: `padding: 20px` (all sides equal).
 - Two values: `margin: 0 5px` (top/bottom, left/right).
 - **Borders:** Defined by **width** (e.g., `5px`), **style** (e.g., `solid`, `dashed`), **color** (e.g., `blue`).
- **Floating:**
 - Moves elements horizontally (`float: left` or `right`), removing them from normal vertical stacking (normal flow).
 - Block elements stack vertically in normal flow; inline elements flow horizontally within blocks.
 - **Clear:** Restores normal flow (`clear: left`, `right`, `both`), prevents overlap with floated elements.
 - Example: A tall floated element may overlap content without `clear: both` to fix layout.
- **Positioning:**
 - **Relative Positioning:** Element stays in normal flow but offset from its original position (e.g., `position: relative; left: 20px; top: 20px;`).
 - Other elements respect its original space, though it may overlap them.
 - **Absolute Positioning:** Removes element from normal flow, positions relative to nearest positioned ancestor or browser window (e.g., `position: absolute; left: 20px; top: 20px;`).
 - Other elements ignore its original space.
 - **Fixed Positioning:** A subtype of absolute, fixes element in the browser window during scrolling (e.g., `position: fixed;`).
 - Not supported in IE5/IE6 without JavaScript workarounds.
- **Web Application Architecture:**
 - **Fundamentals:** A structure linking server and user, acting as the "skeleton" of web apps, responding to user requests.
 - **How It Works:**
 - User enters a URL, generating a request.
 - Browser uses DNS to find the server's IP address.
 - Server retrieves data, sends it back, and the browser displays the page.
 - **Components:**

- **Client-Side (Frontend):** HTML, CSS, JavaScript for user interaction, stored in the browser.
 - **Server-Side (Backend):** PHP, Python, Ruby for HTTP processing in the background.
 - **Structural:** Browser, web app server, database server for functionality.
 - **User Interface:** Visual elements (dashboards, widgets, settings) enhancing UX, not tied to core architecture.
 - **Layered Architecture:**
 - **Presentation Layer:** Manages UI and browser communication (HTML, CSS, JavaScript).
 - **Business Layer:** Processes requests, applies business logic/rules.
 - **Data Access Layer:** Handles CRUD operations (Create, Read, Update, Delete) with data sources (e.g., XML, databases).
 - **Data Service Layer:** Secures data, separates business logic from client-side.
-

Chapter3 - JavaScript

- **Introduction:**
 - JavaScript is a lightweight, interpreted programming language.
 - Easy to implement due to seamless integration with HTML.
 - Open and cross-platform, supported across all modern web browsers (e.g., Chrome, Firefox, Safari).
 - Requires no special environment setup as it's pre-installed in browsers.
 - Versatile for developing both front-end (e.g., jQuery, React) and back-end (e.g., Express, Node.js) software.
 - Originally used in browsers, now extended to mobile app development, desktop apps, and game development.
- **Applications of JavaScript:**
 - **Client-Side Validation:** Verifies user inputs before server submission (e.g., checking form data).
 - **Manipulating HTML Pages:** Dynamically adds, deletes, or modifies HTML tags and styles (e.g., changing look and feel).
 - **User Notifications:** Displays alerts or messages to users.
 - **Back-End Data Loading:** Retrieves server data using techniques like Ajax.
 - **Presentations:** Creates website-like presentations with libraries such as RevealJS or BespokeJS.
 - **Server Applications:** Builds fast, scalable network applications (e.g., web servers) using Node.js.
- **JavaScript Syntax:**
 - Code is placed within `<script>` tags:

```
<html>
  <body>
    <script language="javascript" type="text/javascript">
      document.write("Hello World!")
    </script>
  </body>
</html>
```

- `<script>` can be placed anywhere in the HTML, though `<head>` is recommended.
- `language="javascript"` specifies the scripting language; `type="text/javascript"` is preferred in modern HTML.
- `document.write()` outputs strings or HTML to the document.
- No `main()` function/method required.
- Whitespace (spaces, tabs, newlines) is ignored by the interpreter.
- Semicolons are optional if statements are on separate lines.
- **Data Types:**
 - **Primitive Data Types:**
 - **Numbers:** E.g., `123`, `120.50`, `123e5` (scientific notation); all stored as 64-bit floating-point values (no integer vs. float distinction).
 - **Strings:** E.g., `"hello world"` or `'hello world'` (single or double quotes allowed).
 - **Boolean:** `true` or `false`.
 - **Trivial Data Types:**
 - `null`: Represents no value.
 - `undefined`: A variable without a value (e.g., `let car; // undefined`).
 - **Composite Data Types:**
 - **Objects:** Defined with curly braces `{}` (e.g., `person = {firstName: "John", lastName: "Doe"};`).
 - **Arrays:** Defined with square brackets `[]`, support numbered indexing (e.g., `let students = ['Anteneh', 'Daniel']; document.write(students[1]); // "Daniel"`).
- **Identifiers and Reserved Words:**
 - Identifiers (variable/function names):
 - Must start with a letter, underscore (`_`), or dollar sign (`$`).
 - Subsequent characters can be letters, digits, underscores, or dollar signs.
 - Reserved words (keywords) cannot be used as identifiers (e.g., `function`, `if`, `var`, `let`, `const`, `true`, `null`).
 - Partial list of reserved words:
 - `abstract`, `boolean`, `break`, `case`, `catch`, `class`, `const`, `continue`, `debugger`, `default`, `delete`, `do`, `else`, `enum`, `export`, `extends`, `false`, `final`, `finally`, `for`, `function`, `goto`, `if`, `implements`, `import`, `in`, `instanceof`, `int`, `interface`, `let`, `new`, `null`, `package`, `private`, `protected`, `public`, `return`, `static`, `super`, `switch`, `this`, `throw`, `throws`, `true`, `try`, `typeof`, `var`, `void`, `while`, `with`.
- **Variables:**
 - Containers for storing data values, declared using `var`, `let`, or `const`.
 - **Declaration Examples:**
 - `var firstName;`
 - `let middleName;`
 - `const lastName;`
 - Variables are `undefined` until assigned a value using `=` (e.g., `firstName = "Anteneh";`).
 - Can be initialized during declaration (e.g., `var firstName = "Anteneh";`).

- JavaScript is untyped; variables can hold any data type (e.g., `var year = 3; var grade = 3.5;`).
- **Variable Keywords:**
 - **var:**
 - Can be re-declared and updated.
 - Supports hoisting (usable before declaration).
 - No block scope (accessible outside `{}` blocks).
 - **let** (ES6, 2015):
 - Can be updated but not re-declared.
 - Must be declared before use (no hoisting).
 - Has block scope (not accessible outside `{}` blocks).
 - **const** (ES6, 2015):
 - Cannot be updated or re-declared.
 - Has block scope.
 - Must be initialized at declaration.

- **Operators:**

- **Arithmetic:** `+` (add), `-` (subtract), `*` (multiply), `/` (divide), `%` (modulus), `**` (exponentiation), `++` (increment), `--` (decrement).
- **Assignment:** `=`, `+=`, `-=`, `/=`, `%=`, `*=`.
- **Comparison:** `==` (equal), `===` (strict equal), `!=` (not equal), `!==` (strict not equal), `>`, `>=`, `<`, `<=`, `?` (ternary).
- **Logical:** `&&` (AND), `||` (OR), `!` (NOT).
- **Type:** `typeof` (returns type), `instanceof` (checks object type).
- **Bitwise:** `&`, `|`, `~`, `^`, `<<`, `>>`, `>>>`.
- **Examples:**
 - `let x = 5; let y = 2; let mult = x * y; // 10`
 - `y *= x; // y becomes 10`
 - `let power_y = y ** 2; // 4`
 - `document.write(typeof(y)); // "number"`
- **String Operations:**
 - `+` concatenates strings (e.g., `"Anteneh" + " " + "Daniel" // "Anteneh Daniel"`).
 - `+=` appends to strings (e.g., `text1 += " is a Lecturer";`).
 - Number + String = String (e.g., `"10" + 5 // "105"`).
 - Order matters (e.g., `"The result is: " + 10 + 20 // "The result is: 1020"`, but `10 + 20 + " is The Result" // "30 is The Result"`).
- **Numeric Conversion:**
 - JavaScript converts strings to numbers in arithmetic (e.g., `"100" * "2" // 200`).
 - Invalid operations yield `NaN` (e.g., `100 / "Apple" // NaN`).

- **Conditional Statements:**

- Used to perform actions based on conditions.
- **If Statement:**
 - Forms: `if`, `if...else`, `if...else if...else`.
 - Examples:

```

var age = 20;
if (age > 18) {
    document.write("<b>Qualifies for driving</b>");
}
var age = 15;
if (age > 18) {
    document.write("<b>Qualifies for driving</b>");
} else {
    document.write("<b>Does not qualify for driving</b>");
}
var book = "maths";
if (book == "history") {
    document.write("<b>History Book</b>");
} else if (book == "maths") {
    document.write("<b>Maths Book</b>");
} else {
    document.write("<b>Unknown Book</b>");
}

```

- **Switch Statement:**

- Handles multiway branching based on a single variable's value.
- Useful for multiple conditions (syntax not fully provided in document).

- **Loops:**

- **For Loop:**

- Syntax: `for (initialization; test condition; iteration) { code }.`
- Example:

```

for (count = 0; count < 10; count++) {
    document.write("<li> Current Count : " + count + "</li>");
}

```

- **For/In Loop:**

- Loops through object properties or array indices.
- Example:

```

let person = {firstName: 'anteneh', age: 12};
for (var_name in person) {
    document.write(var_name); // property name
    document.write(person[var_name]); // property value
}
let students = ['Anteneh', 'daniel'];
for (var_name in students) {
    document.write(students[var_name]); // array value
}

```

- **For/Of Loop:**

- Loops through values of iterable objects (e.g., arrays, strings).
- Example:

```
let students = ['Anteneh', 'daniel'];
for (var_name of students) {
  document.write(var_name); // value
}
```

- **While Loop:**

- Syntax: `while (expression) { code }`.
- Example:

```
var count = 0;
while (count < 10) {
  document.write("<li> Current Count : " + count + "</li>");
  count++;
}
```

- **Do/While Loop:**

- Executes at least once, then loops while condition is true.
- Example:

```
var count = 0;
do {
  document.write("<li> Current Count : " + count + "</li>");
  count++;
} while (count < 10);
```

- **Functions:**

- Blocks of code designed for specific tasks, invoked when called.
- Declared with `function` keyword, no return type specified.
- Example:

```
function add(num1, num2) {
  return num1 + num2;
}
var sum = add(12, 34); // 46
```

- **Example Code (Guessing Game):**

- Demonstrates variables, loops, and conditionals:

```
var thinkings = Math.ceil(Math.random() * 1000); // Random number 1-1000
var guess = prompt("I'm thinking of a number between 1 and 1000. What is it?", " ");
while (guess != thinkings) {
    if (guess < thinkings) {
        guess = prompt("Your guess was too low. Guess again.", " ");
    } else {
        guess = prompt("Your guess was too high. Guess again.", " ");
    }
}
window.alert(guess + " is correct!");
```

- **Where to Place JavaScript:**

- Inserted between `<script>` and `</script>` tags.
- Options:
 - **<head>**: Common for initialization scripts.
 - **<body>**: Often at end for faster page loading.
 - **External Files**: `<script src="script.js"></script>`.
- **External Advantages:**
 - Separates HTML and code.
 - Improves readability and maintenance.
 - Cached files speed up page loads.

AJAX (Asynchronous JavaScript and XML)

- **Definition:**

- AJAX is a technique for creating fast and dynamic web pages by allowing data exchange with a server without reloading the entire page.
- Combines JavaScript, XML (or JSON), and asynchronous communication to update parts of a webpage seamlessly.

- **Purpose:**

- Enables asynchronous data retrieval from a server, improving user experience by avoiding full page refreshes.
- Commonly used for loading content dynamically (e.g., updating a news feed, fetching search results).

- **Core Components:**

- **JavaScript**: Handles client-side logic and DOM manipulation.
- **XMLHttpRequest Object**: Facilitates communication with the server (modern alternatives include the Fetch API).
- **Server-Side Resource**: Provides data (e.g., a PHP script, API endpoint returning JSON or XML).

- **Asynchronous Processing:** Allows the browser to continue functioning while waiting for server responses.
- **How AJAX Works:**
 - User triggers an event (e.g., clicking a button).
 - JavaScript creates an `XMLHttpRequest` object and sends a request to the server.
 - Server processes the request and sends back data (e.g., in XML, JSON, or plain text).
 - JavaScript receives the response and updates the webpage dynamically (e.g., via `innerHTML`).
- **Basic Syntax with XMLHttpRequest:**
 - Example:

```
// Step 1: Create XMLHttpRequest object
let xhr = new XMLHttpRequest();

// Step 2: Define callback for response
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4 && xhr.status == 200) {
    // Step 4: Handle response
    document.getElementById("result").innerHTML = xhr.responseText;
  }
};

// Step 3: Open and send request
xhr.open("GET", "server-script.php", true); // true = asynchronous
xhr.send();
```

- **Key Methods:**
 - `open(method, url, async)`: Initializes request (e.g., `GET` or `POST`, target URL, async flag).
 - `send()`: Sends the request to the server (can include data for `POST`).
- **Properties:**
 - `readyState`: Tracks request status (0 = uninitialized, 1 = loading, 2 = loaded, 3 = interactive, 4 = complete).
 - `status`: HTTP status code (e.g., 200 = OK, 404 = Not Found).
 - `responseText`: Server response as a string.
 - `responseXML`: Server response as an XML document (if applicable).
- **AJAX with POST Example:**
 - Sending data to the server:

```
let xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4 && xhr.status == 200) {
    document.getElementById("result").innerHTML = xhr.responseText;
  }
};
```

```
xhr.open("POST", "server-script.php", true);
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhr.send("name=John&age=30");
```

- `setRequestHeader()`: Specifies the content type for POST data.

- **Advantages:**

- Improves performance by updating only specific page sections.
- Enhances interactivity (e.g., real-time form validation, live search suggestions).
- Reduces server load and bandwidth usage compared to full page reloads.

- **Disadvantages:**

- Browser back button may not work as expected (requires additional handling).
- Increases complexity in JavaScript code.
- Relies on JavaScript being enabled in the browser.

- **Modern Alternative: Fetch API:**

- A simpler, promise-based alternative to `XMLHttpRequest`.
- Example:

```
fetch("server-script.php")
  .then(response => response.text())
  .then(data => {
    document.getElementById("result").innerHTML = data;
  })
  .catch(error => console.log("Error:", error));
```

- Supports `async/await` for cleaner syntax:

```
async function getData() {
  try {
    let response = await fetch("server-script.php");
    let data = await response.text();
    document.getElementById("result").innerHTML = data;
  } catch (error) {
    console.log("Error:", error);
  }
}
getData();
```

- **Common Use Cases:**

- Loading data in the background (e.g., fetching user posts without refreshing).
- Submitting forms asynchronously (e.g., saving settings instantly).

- Real-time updates (e.g., chat messages, stock prices).

- **Data Formats:**

- Traditionally used XML, but JSON is now more common due to its lightweight nature and easy parsing in JavaScript.
- Example with JSON:

```
xhr.onreadystatechange = function() {  
  if (xhr.readyState == 4 && xhr.status == 200) {  
    let data = JSON.parse(xhr.responseText);  
    document.getElementById("result").innerHTML = data.name;  
  }  
};  
xhr.open("GET", "server-script.json", true);  
xhr.send();
```

- **Error Handling:**

- Check `status` for server errors (e.g., `404`, `500`).
- Use `onerror` or try-catch with Fetch to handle network issues.
- Example:

```
xhr.onerror = function() {  
  console.log("Request failed");  
};
```
