Difference between

i) HTML and HTML5

Feature	HTML	HTML5
Version	Original version (static HTML)	Fifth and latest version of HTML
Doctype Declaration	Long and complex (`<!DOCTYPE HTML PUBLIC ...>`)	Simplified declaration (`<!DOCTYPE html>`)
Multimedia Support	Requires external plugins (e.g., Flash for videos)	Native support for audio and video (`<audio>`, `<video>`)
Semantics	Limited semantic tags like `<div>`	New semantic elements like `<header>`, `<footer>`, `<section>`, `<article>`
Form Elements	Basic elements (`<input>`, `<textarea>`)	New input types (e.g., `date`, `email`, `range`) and attributes (e.g., `required`, `placeholder`)
Graphics	No built-in graphics support	Supports `<canvas>` and Scalable Vector Graphics (SVG)
Offline Capability	No offline support	Provides offline web application support using **AppCache** and later **Service Workers**
Geolocation API	Not available	Built-in Geolocation API for location tracking
Browser Compatibility	Supported by older browsers	Requires modern browsers for full feature support
Drag and Drop	Not supported	Native drag-and-drop API support
Performance	Relatively less optimized	Improved performance for multimedia and scripting
Storage	Uses cookies for client-side storage	Supports localStorage and sessionStorage APIs
Accessibility	Limited accessibility features	Enhanced accessibility with ARIA (Accessible Rich Internet Applications) attributes

ii) Class selector and ID selector

Feature	Class Selector (`.`)	ID Selector (#)
Symbol	Denoted by a period (`.`)	Denoted by a hash (#)
Uniqueness	Can be reused multiple times across elements	Must be unique within a document
Usage	Targets multiple elements with the same class	Targets a single, unique element
Syntax	`.class-name { ... }`	`#id-name { ... }`
Priority	Lower specificity compared to ID selectors	Higher specificity than class selectors
Purpose	Used for grouping similar elements for styling	Used to uniquely identify and style a specific element
Example in HTML	`<div class="box"></div>`	`<div id="unique-box"></div>`
Example in CSS	`.box { color: red; }`	`#unique-box { color: blue; }`
Best Practice	Use for elements sharing common styles	Use for one-of-a-kind elements (e.g., logo, main header)
Flexibility	More flexible, supports multiple classes per element	Less flexible, only one ID can be assigned to an element

iii)Get method and POST method in PHP

Aspect	GET Method	POST Method
Purpose	Used to retrieve data from the server.	Used to send data to the server for processing.
Data Visibility	Appends data in the URL, making it visible.	Data is included in the request body, hidden from URL.
Security	Less secure as data is exposed in the URL.	More secure as data is not displayed in the URL.
Data Limit	Limited to URL length (varies by browser).	No restriction on the amount of data sent.
Bookmarkable	Can be bookmarked as the data is part of the URL.	Cannot be bookmarked as data is not stored in the URL.
Use Case	Suitable for fetching or reading data (e.g., search queries).	Suitable for submitting sensitive or large data (e.g., forms, uploads).
Syntax in PHP	Access data using `$_GET['key']`.	Access data using `$_POST['key']`.
Example in Form	`<form method="GET" action="process.php">`	`<form method="POST" action="process.php">`
Caching	Can be cached by browsers.	Not cached by default.
Encoding	Sends data as URL-encoded key-value pairs.	Sends data in the request body, can be encoded in multiple formats.
Debugging	Easier to debug as data is visible in the URL.	Requires tools like Postman or browser developer tools for debugging.

iv)JSON and XML

Aspect	JSON (JavaScript Object Notation)	XML (eXtensible Markup Language)
Structure	Lightweight, uses key-value pairs	Heavier, uses nested tags
Readability	Human-readable and concise	Human-readable but verbose
Syntax	Uses curly braces `{}`, square brackets `[]`, and key-value pairs	Uses opening and closing tags `<tag>` and `</tag>`
Data Type Support	Supports native types like strings, numbers, arrays, booleans, and null	Represents all data as text, requiring conversion for complex types
Schema Validation	No strict schema required	Schema can be enforced with DTD or XSD
Parsing Speed	Faster to parse due to lightweight nature	Slower to parse because of verbosity
Usage	Primarily used in web APIs, modern applications	Used in legacy systems, SOAP web services
Self-descriptive	Partially (requires keys to describe data)	Fully self-descriptive with tags
Extensibility	Limited flexibility for metadata	Highly extensible with attributes and nested tags
Interoperability	Best for web-based applications and APIs	Widely used across various platforms
Example	`{ "name": "John", "age": 30 }`	`<person><name>John</name><age>30</age></person>`
File Size	Smaller file size	Larger file size due to tag overhead
Support	Supported natively in JavaScript, Python, etc.	Supported in a wide range of languages and tools
Error Handling	Limited error messages in case of invalid syntax	Provides detailed error reporting for invalid documents

v) Internal DTD and External DTD

Aspect	Internal DTD	External DTD
Definition	Defined within the same XML document.	Defined in a separate file, referenced by the XML document.
Syntax Location	Declared inside the `<!DOCTYPE>` section of the XML document.	Declared in a separate `.dtd` file and linked to the XML document.
Usage Syntax	`<!DOCTYPE root-element [ ... DTD rules ... ]>`	`<!DOCTYPE root-element SYSTEM "file.dtd">` (or `PUBLIC`)
Accessibility	Self-contained, making the document portable.	Requires external file access, less portable.
Maintenance	Harder to maintain for large or complex documents, as DTD rules are embedded.	Easier to maintain, as the DTD is stored in one location and can be reused.
Reusability	Cannot be reused across multiple XML documents.	Can be reused for multiple XML documents.
Performance	Faster processing as no external file is loaded.	Slightly slower as it involves loading an external file.
Best Use Case	Suitable for small, standalone XML documents.	Suitable for large projects with multiple XML documents sharing the same structure.

vi)Ajax application model and Traditional application model

Aspect	Ajax Application Model	Traditional Application Model
Communication	Asynchronous communication with the server.	Synchronous communication with the server.
Page Reload	Updates parts of the page dynamically without reloading the entire page.	Requires full page reloads for updates or data changes.
Speed	Faster as only the required data is fetched and updated.	Slower as the entire page needs to reload on every interaction.
User Experience	Provides a more interactive and seamless user experience.	Offers a less interactive experience with noticeable page reloads.
Network Load	Reduces network load by fetching only necessary data.	Increases network load as the entire page is reloaded.
Technology	Relies on JavaScript, XML/JSON, and XMLHttpRequest for dynamic updates.	Relies on traditional server-side rendering and HTTP requests.
Responsiveness	Highly responsive and suitable for real-time updates.	Less responsive, as updates are delayed by page reloads.
Data Transfer	Transfers minimal data (e.g., JSON or XML).	Transfers entire HTML pages.
Example Use Case	Applications like Gmail, Google Maps, or Facebook.	Static websites or early web applications.
Development Complexity	More complex due to asynchronous processing and handling dynamic updates.	Simpler as it follows a straightforward request-response cycle.
Browser Compatibility	Requires modern browser support for JavaScript and AJAX features.	Compatible with older browsers but lacks modern features.