



KAMALDNP

kamaldnp.com • IT Training, Varanasi

PDF created by Raj Kamal using AI

Short Notes — Web Technologies

BTEUP Diploma in Computer Science & Engineering • 3rd Semester • Short Notes
Session 2026-27

Short Revision Notes — prepared by KAMALDNP Academy from the official BTEUP 2026-27 syllabus. This is NOT a BTEUP board paper or publication.

These **short revision notes** cover all five units of Web Technologies from the official BTEUP 2026-27 syllabus for CSE 3rd semester, in syllabus order. Topics are clubbed into sensible point entries, and tiny correct code snippets are given where a five-mark answer needs them.

Please note: Web Technologies is a **practicum** subject in the 2026-27 scheme, so these notes target its **theory (lecture) part and viva** — use them together with your practical file, the official syllabus PDF and the matching practice paper. These are revision notes only, not a replacement for coding practice and class notes.

Unit 1 — Web Development Introduction (2 Periods, 10%)

Topics: WWW, internet, browser, search engine; client-server model; URL; web pages, website and web services; types of websites (static, dynamic, responsive); developer options of browser (view page source, developer tools, inspect element).

WWW, internet, browser, search engine

Internet is the worldwide network of networks that connects millions of computers using standard protocols (TCP/IP). It carries many services — web, email, file transfer and chat. The **World Wide Web (WWW)** is only one service of the internet: it is the system of interlinked hypertext documents (web pages) accessed through browsers using HTTP/HTTPS. So: the internet is the road network; the web is one kind of traffic on it.

A **web browser** is client software that requests web pages from servers, interprets their HTML, CSS and JavaScript, and displays them — examples: Chrome, Firefox, Edge. A **search engine** is a website that helps users find pages: it crawls pages in advance, stores them in an index, and on a query returns a ranked list of links — examples: Google, Bing. A browser is a tool for viewing pages; a search engine is a tool for finding pages, and it is itself opened inside a browser.

Client-server model

In the **client-server model**, work is divided between two roles. The **client** (normally the user's browser) requests a service; the **server** (a computer running web-server software) listens continuously, processes the request and sends back a response. One server serves many clients.

Steps when a page is opened:

- The browser (client) takes the URL and finds the server's IP address using DNS.
- It sends an HTTP request naming the page or resource wanted.
- The server reads the requested file, or runs a server-side program (such as PHP) to build the page.
- It sends an HTTP response containing HTML (plus CSS/JavaScript/images).
- The browser renders the page; further requests fetch the linked files.

Advantages: data and processing are centralised and easy to maintain, security is controlled at the server, and many users can share the same application. This request-response pattern is the base of everything else in this subject.

URL

A **URL (Uniform Resource Locator)** is the unique address of a resource on the web — the text typed in the browser's address bar. Its general form is:

scheme://host:port/path?query#fragment

Parts, with the example <https://www.kamaldnp.com:443/courses/list.php?sem=3#top>:

- **Scheme / protocol** — https — how to fetch the resource (http, https, ftp).
- **Host (domain name)** — www.kamaldnp.com — the server's name, converted to an IP address by DNS.
- **Port** — 443 — optional; default is 80 for http and 443 for https, so it is usually omitted.
- **Path** — /courses/list.php — the folder and file on the server.
- **Query string** — ?sem=3 — extra data passed to the page as name=value pairs joined by &.
- **Fragment** — #top — a position inside the page; it is handled by the browser and not sent to the server.

Remember: a URL is one type of URI, specialised in locating a resource.

Web pages, website and web services

A **web page** is a single document on the web, written mainly in HTML and viewed in a browser; it may contain text, images, links, audio and video, and has its own URL. The first page of a site is its **home page**.

A **website** is a collection of related web pages of one person or organisation, stored on a server under one domain name — for example, all pages under kamaldnp.com together form one website. Pages are joined by hyperlinks so a visitor can move between them.

A **web service** is different: it is a service that one software offers to another software over the web using standard formats (commonly exchanging data as JSON or XML over HTTP), rather than pages for humans to read. Example: a weather app on a phone gets its data from a weather web service; a payment gateway is also used as a web service. In one line: web pages are for people, web services are for programs.

Types of websites (static, dynamic, responsive)

- **Static website** — pages are fixed HTML files stored exactly as they will be shown; every visitor sees the same content, and content changes only when the developer edits and re-uploads the files. Fast, cheap and secure; suitable for a small information or portfolio site. No database is used.
- **Dynamic website** — pages are generated at request time by server-side programs (such as PHP) using a database, so content can differ per user and be updated through an admin panel without editing code. Examples: news portals, e-commerce and social sites. More powerful, but slower and costlier to build and maintain.
- **Responsive website** — not a separate content type but a design quality: the layout automatically adjusts to the size of the screen (mobile, tablet, desktop) using flexible grids, flexible images and CSS media queries, so no separate mobile site is needed. A modern site is usually both dynamic and responsive.

Exam tip: static vs dynamic is about **how content is produced**; responsive is about **how the page adapts to the device**.

Developer options of browser (view page source, developer tools, inspect element)

Modern browsers include built-in tools for viewing and testing how a page is built:

- **View Page Source** (commonly Ctrl+U) — shows the exact HTML code received from the server for the page, in a read-only view. Useful for learning structure and checking tags, but it shows the original source, not changes made later by JavaScript.
- **Developer Tools (DevTools)** (commonly F12) — a full panel set: Elements (live HTML/CSS), Console (JavaScript output and errors, and running commands), Network (files loaded, their size and time), Sources and Application, among others. Changes made here are temporary experiments on your own screen and disappear on refresh.

- **Inspect Element** (right-click on any item → Inspect) — opens DevTools with exactly that element selected, showing its HTML and the CSS rules applied to it; you can edit a value live to preview the effect.

These tools are essential for debugging web practicals: most layout and JavaScript errors are first located here.

Likely question: Differentiate between a static and a dynamic website. Explain the parts of a URL with an example, and describe the working of the client-server model.

Unit 2 — HTML Basics (3 Periods, 25%)

Topics: HTML document and basic structure; syntax; HTML tags and attributes; types of tags; rules of nesting; basic tags; page formatting (paragraph, line break, blank space, background, div and span); text formatting (headings, bold, italic, marked, small, deleted, inserted, subscript, superscript); comments; horizontal lines; lists (ordered, unordered, definition); images, text links, image links, opening a page in a new window/tab, linking within a page; table tags, frames and iframe; HTML forms.

HTML document and basic structure; syntax

HTML (HyperText Markup Language) is the standard language for creating web pages. It is not a programming language — it marks up content to tell the browser how to structure it. HTML files are saved with .html or .htm extension. Basic structure of every document:

```
<!DOCTYPE html>
<html>
<head>
<title>My Page</title>
</head>
<body>
Visible content here
</body>
</html>
```

- **<!DOCTYPE html>** — declares that the document is HTML5; it is not a tag.
- **<html>** — the root element containing everything.
- **<head>** — information about the page (title, meta data, links to CSS); not displayed.
- **<title>** — text shown on the browser tab.
- **<body>** — everything the visitor actually sees.

Syntax rule: content is written between an opening tag and a closing tag, like `<p>text</p>`; tag names are not case-sensitive, but lowercase is the standard practice.

HTML tags and attributes; types of tags; rules of nesting

A **tag** is a keyword inside angle brackets that instructs the browser, such as `<p>` for a paragraph. An **attribute** gives extra information about an element and is written inside the opening tag as `name="value"`, for example `<p align="center">` or ``.

Types of tags:

- **Container (paired) tags** — have opening and closing tags: `<p>...</p>`, `<div>...</div>`.
- **Empty (self-closing) tags** — have no closing tag and no content: `<br>`, `<hr>`, `<img>`.
- **Block-level tags** — start on a new line and take full width: `<p>`, `<div>`, `<h1>`.
- **Inline tags** — stay within the line: `<span>`, `<b>`, `<a>`.

Rules of nesting: tags opened inside another tag must close before it — last opened, first closed. Correct: `<p><b>text</b></p>`. Wrong (overlapping): `<p><b>text</p></b>`. Proper nesting keeps the document valid and the display predictable.

Basic tags and page formatting (paragraph, line break, blank space, background, div and span)

Core tags for building and formatting a page:

`<p>This is a paragraph.</p>`

Text with a line break `<br>` on the next line.

`<body style="background-color:lightyellow;">`

`<div>A block section of the page</div>`

`<span>A small inline part</span>`

- **<p>** — paragraph; the browser adds space before and after it.
- **
** — line break; starts the text after it on a new line without starting a new paragraph.
- **Blank space** — the browser collapses extra spaces and Enter presses into one space; use ` ` (non-breaking space) for a forced blank space and `<pre>` to show text exactly as typed.
- **Background** — set by the `bgcolor` attribute in older style, or by CSS `background-color` / `background-image` on the `body` or any element.
- **<div>** — a block-level division used to group a section for layout and styling.
- **** — an inline container used to style a word or phrase inside a line.

Text formatting (headings, bold, italic, marked, small, deleted, inserted, subscript, superscript)

HTML gives ready-made tags for the appearance and meaning of text:

`<h1>Main Heading</h1>` ... `<h6>Smallest Heading</h6>`

`<b>Bold</b>` `<i>Italic</i>` `<mark>Marked</mark>`

`<small>Small text</small>` `<del>Deleted</del>` `<ins>Inserted</ins>`
H`<sub>2</sub>`O and x`<sup>2</sup>`

- **Headings `<h1>` to `<h6>`** — six levels; h1 is the largest and most important, h6 the smallest. Use them for structure, not just for size.
- **`<b>` bold and `<i>` italic** — visual emphasis; `<strong>` and `<em>` carry the same look with added meaning.
- **`<mark>`** — highlights text as if with a marker.
- **`<small>`** — smaller text, used for notes.
- **`<del>`** — struck-through deleted text; **`<ins>`** — underlined inserted text; together they show edits in a document.
- **`<sub>`** subscript (H₂O) and **`<sup>`** superscript (x squared) — for formulas.

Exam tip: present this topic as a small table — tag in the first column, its effect in the second and one short example in the third; one line per tag is exactly what a five-mark answer needs.

Comments and horizontal lines

An HTML **comment** is a note for the developer written inside the code; the browser ignores it completely, so it never appears on the page.

Syntax: `<!-- This is a comment -->`

- Uses: explaining a section of code, noting who wrote it and when, and temporarily hiding code while testing (commenting out) instead of deleting it.
- A comment can span several lines, but it cannot be nested inside another comment.
- Warning for exams and practice: comments are visible to anyone who views the page source, so never write passwords or private data in them.

A **horizontal line** is drawn by the empty tag `<hr>`. It separates sections of a page visually. Its look can be controlled by attributes in simple code or, better, by CSS:

```
<hr>
<hr style="width:50%;">
```

Typical use: between the header and content, or between two topics on one page.

Lists in HTML (ordered, unordered, definition)

Lists present related items in a clean form. HTML has three list types:

1. Ordered list — items are numbered (1, 2, 3 or a, b, c or i, ii, iii using the type attribute):

```
<ol>
<li>First step</li>
<li>Second step</li>
</ol>
```

2. Unordered list — items carry bullets (disc, circle or square by the type attribute):

```
<ul>
<li>Tea</li>
<li>Coffee</li>
</ul>
```

3. Definition list — for term-meaning pairs, using <dl> for the list, <dt> for the term and <dd> for its description:

```
<dl>
<dt>HTML</dt>
<dd>Language for web pages.</dd>
</dl>
```

- Every item of ordered and unordered lists is written with ****.
- Lists can be **nested** — a list inside an makes a sub-list, commonly used for menus.
- Choose by meaning: sequence matters → ordered; only a set of items → unordered; glossary → definition list.

Images and links (images, text links, image links, new window/tab, linking within a page)

Images are inserted with the empty tag . Important attributes: src (image file), alt (alternative text shown if the image cannot load, and read by screen readers), width and height.

```

```

Links are created by the anchor tag <a> with the href attribute giving the destination:

```
<a href="courses.html">Our Courses</a> (text link)
<a href="courses.html"></a> (image link — the image
becomes clickable)
<a href="courses.html" target="_blank">Open in a new tab</a>
<a href="#fees">Jump to Fees</a> with <h2 id="fees">Fees</h2>
```

- **target="_blank"** — opens the linked page in a new window/tab; without it the page opens in the same tab.
- **Linking within a page** — give the target element an id, and link to it with href="#id"; clicking scrolls to that position. Used for long pages and “back to top” links.
- Links may also start an email (href="mailto:...") or a download.

Table tags, frames and iframe

An HTML **table** shows data in rows and columns. Main tags:

```
<table border="1">
<tr><th>Roll</th><th>Name</th></tr>
```

```
<tr><td>101</td><td>Aarav</td></tr>
</table>
```

- **<table>** starts the table; **<tr>** makes a row; **<th>** makes a bold, centred heading cell; **<td>** makes a normal data cell. Attributes `rowspan` and `colspan` merge cells vertically and horizontally; `<caption>` gives the table a title.

Frames (old method, `<frameset>` with `<frame>`) divided the browser window into independent sections, each showing a different page. They are **obsolete in HTML5** — difficult to bookmark, print and use on mobiles — so they should be named only as a historical topic.

An **iframe** (inline frame) is the modern, supported way to embed one page inside another:

```
<iframe src="map.html" width="400" height="300"></iframe>
```

Common uses: embedding a map, a video or another site's widget inside your page.

HTML forms

An HTML **form** collects input from the visitor and sends it to a server program for processing (login, registration, search, feedback). Structure:

```
<form action="save.php" method="post">
Name: <input type="text" name="sname">
```

```
Password: <input type="password" name="pwd">
```

```
Gender: <input type="radio" name="g" value="M"> Male
<input type="radio" name="g" value="F"> Female
```

```
Hobbies: <input type="checkbox" name="h1"> Music
```

```
<input type="submit" value="Register">
<input type="reset" value="Clear">
</form>
```

- **action** — the server page that will receive the data; **method** — get (data visible in the URL, small amounts) or post (data hidden in the request body, used for forms and passwords).
- **Input types** — text, password (characters hidden), radio (choose one of a group sharing one name), checkbox (choose many), submit, reset; also `<textarea>` for multi-line text and `<select>` for a drop-down list.
- The **name** attribute is essential — the server receives data by these names.

Likely question: Write HTML code for a student registration form with name, password, gender (radio), hobbies (checkbox) and submit/reset buttons. Also explain ordered, unordered and definition lists with examples.

Unit 3 — Cascading Style Sheets (3 Periods, 15%)

Topics: Introduction and benefits of CSS; CSS syntax; CSS implementation (inline, internal, external); CSS selectors (ID, class, grouping, universal, pseudo-classes); CSS properties (background colour/image, border style, height, width, colour, text-align, font family/style/size, font-weight); box model in CSS (margin, border, padding).

Introduction and benefits of CSS; CSS syntax

CSS (Cascading Style Sheets) is the language that controls the presentation of a web page — colours, fonts, sizes, spacing and layout — while HTML provides the structure and content. “Cascading” means that when several rules apply to one element, they are combined by a fixed priority order.

Benefits:

- Content and design are separate, so pages are cleaner and easier to maintain.
- One style file can design a whole website; changing one rule updates every page.
- Pages load faster because style code is not repeated in every page.
- The same HTML can be shown differently on mobile, tablet and print.
- Designs become consistent across the site.

CSS syntax: a rule has a **selector** and a declaration block of property:value pairs:

```
p { color: blue; font-size: 16px; }
```

Here p is the selector, color and font-size are properties, and blue and 16px are their values. Declarations are separated by semicolons, and /* this */ is a CSS comment.

CSS implementation (inline, internal, external)

CSS can be added to a page in three ways:

1. Inline CSS — written in the style attribute of a single element:

```
<p style="color:red;">Text</p>
```

Applies to that element only; has the highest priority, but mixes design with content and cannot be reused — use for rare, one-off cases.

2. Internal (embedded) CSS — written in a <style> block inside the <head> of the same page:

```
<style> p { color: blue; } </style>
```

Applies to that one page only; suitable for a single-page document or a page needing special styles.

3. External CSS — rules are kept in a separate .css file and linked to any number of pages:

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

This is the recommended method for real websites: one change in the file restyles the whole site, and the browser caches the file for speed.

Priority (cascade) order: inline style beats internal and external rules; between internal and external, the one written later wins; a more specific selector beats a general one.

CSS selectors (ID, class, grouping, universal, pseudo-classes)

A **selector** chooses the element(s) a rule will style. Element selector (p) styles all paragraphs. The five syllabus selectors:

```
p { color: black; } /* element */
#head { color: navy; } /* ID */
.note { color: green; } /* class */
h1, h2, p { margin: 5px; } /* grouping */
* { font-family: Arial; } /* universal */
a:hover { color: red; } /* pseudo-class */
```

- **ID selector (#)** — styles the one element with that unique id, for example `<h1 id="head">`. An id must be used only once per page.
- **Class selector (.)** — styles all elements sharing that class, for example `<p class="note">`. Classes are reusable many times.
- **Grouping selector** — several selectors separated by commas share one declaration block, avoiding repetition.
- **Universal selector (*)** — matches every element on the page.
- **Pseudo-classes** — select an element in a special state, written with a colon: `a:hover` (mouse over a link), `a:visited`, `input:focus`, `p:first-child`.

CSS properties (background, border, size, colour, text and font)

The syllabus properties, with a combined example:

```
div {
background-color: lightyellow;
background-image: url("bg.png");
border-style: solid;
height: 200px;
width: 50%;
color: darkblue;
text-align: center;
font-family: Arial, sans-serif;
font-style: italic;
```

```
font-size: 18px;
font-weight: bold;
}
```

- **background-color** — background colour of an element; **background-image** — background picture (url of the image).
- **border-style** — style of the border: solid, dashed, dotted, double, none (width and colour have their own properties).
- **height / width** — size of the element, in px, % or other units.
- **color** — colour of the text.
- **text-align** — left, right, center or justify.
- **font-family** — the typeface list (first available is used); **font-style** — normal or italic; **font-size** — size of text; **font-weight** — normal, bold or a number such as 700.

Colour values may be names (red), hex codes (#ff0000) or rgb() values — all three forms are acceptable in answers if used correctly.

Box model in CSS (margin, border, padding)

CSS treats every element as a rectangular **box**. The **box model** describes the layers of that box, from inside to outside:

- **Content** — the actual text or image, sized by width and height properties.
- **Padding** — empty space **inside** the box, between the content and the border. It takes the element's background colour. Example: padding: 10px;
- **Border** — the line around the padding, set by border-style, border-width and border-color. Example: border: 2px solid black;
- **Margin** — empty space **outside** the box, between this element and neighbouring elements. It is always transparent. Example: margin: 15px;

So the real space an element occupies is: total width = margin + border + padding + content width + padding + border + margin (left and right together), and similarly for height.

```
div { width: 300px; padding: 10px; border: 2px solid black; margin: 15px; }
```

Diagram in words: draw four nested rectangles and label them — content in the centre, then padding, then border, then margin outside. Common confusion to avoid: padding is inside the border, margin is outside it.

Likely question: Explain the CSS box model with a diagram. Write CSS to style all paragraphs with blue colour and 16px Arial font using an external style sheet, and differentiate between ID and class selectors.

Unit 4 — JavaScript (3 Periods, 25%)

Topics: JavaScript introduction; variables, data types, operators; control flow (if-else, for loop, while loop, do-while loop); declaring functions and calling functions with parameters; adding JavaScript to web documents; JavaScript objects; Document Object Model; HTML events and calling JavaScript functions on events.

JavaScript introduction; variables, data types, operators

JavaScript is a scripting language that runs mainly in the browser (client-side) and makes web pages interactive and dynamic — validating forms, responding to clicks, changing content without reloading. It is interpreted, loosely typed and is a different language from Java despite the similar name.

Variables store data and are declared with var, let or const:

```
let name = "Aarav";  
let marks = 85;  
const PI = 3.14;
```

Data types: number (85, 3.14), string (text in quotes), boolean (true/false), undefined (declared but no value), null (intentionally empty), and object (collections, covered later).

Operators:

- **Arithmetic** — + − * / % and ++ --; note that + also joins strings.
- **Assignment** — = and combined forms like += .
- **Comparison** — == , === (equal value and type), != , > , < , >= , <= ; comparisons return true or false.
- **Logical** — && (AND), || (OR), ! (NOT), used to combine conditions.

Output in practice: document.write(), alert() and console.log().

Control flow (if-else, for loop, while loop, do-while loop)

Control flow statements decide which code runs and how many times.

if-else — runs one block when a condition is true and another when false:

```
if (marks >= 40) {  
  document.write("Pass");  
} else {  
  document.write("Fail");  
}
```

for loop — used when the number of repetitions is known; initialisation, condition and update are in one line:

```
for (let i = 1; i <= 5; i++) {
```

```
document.write(i + "<br>");  
}
```

while loop — checks the condition first, then repeats while it is true; if the condition is false at the start, the body never runs:

```
let i = 1;  
while (i <= 5) { document.write(i); i++; }
```

do-while loop — runs the body first, then checks the condition, so the body executes **at least once**:

```
let j = 1;  
do { document.write(j); j++; } while (j <= 5);
```

Key difference for exams: while is entry-controlled, do-while is exit-controlled. Always update the loop variable inside, otherwise the loop never ends.

Declaring functions and calling functions with parameters

A **function** is a reusable block of code that performs one task. It is written once and **called** (invoked) whenever that task is needed, which avoids repetition and keeps programs organised.

Declaration with parameters:

```
function add(a, b) {  
  let sum = a + b;  
  return sum;  
}
```

Calling with arguments:

```
let result = add(10, 20);  
document.write(result); // prints 30
```

- **Parameters** (a, b) are the variables named in the declaration; **arguments** (10, 20) are the actual values passed in a call. Values are matched by position.
- **return** sends a value back to the caller and ends the function; a function without return simply performs its actions, for example showing an alert.
- Variables declared inside a function with let are **local** to it and are not available outside.

A function can be called from another script statement, from a button click or any other event (last point of this unit), which is how most practical web pages use functions.

Adding JavaScript to web documents

JavaScript can be placed in an HTML document in three ways, parallel to the three ways of adding CSS:

1. Internal (embedded) — code inside a <script> tag in the <head> or <body> of the same page:

```
<script>
alert("Welcome");
</script>
```

The script in the body, just before `</body>`, is commonly used so the page content loads first.

2. External — code kept in a separate .js file and linked:

```
<script src="myscript.js"></script>
```

Best for real sites: the same file serves many pages, keeps HTML clean and is cached by the browser.

3. Inline (event attribute) — a short statement written in an event attribute of an element:

```
<button onclick="alert('Hello')">Click</button>
```

Rules to remember: the `<script>` tag is used in all document-based methods; a script with `src` should not also carry code between its tags; and JavaScript is case-sensitive — `getElementByld` and `getElementbyld` are not the same.

JavaScript objects

An **object** in JavaScript is a collection of related data and actions stored together under one name. Data items are stored as **properties** (name:value pairs) and actions as **methods** (functions belonging to the object).

```
let student = {
  name: "Meera",
  marks: 91,
  show: function() {
    document.write(this.name + " : " + this.marks);
  }
};
```

```
student.show(); // prints Meera : 91
document.write(student.name);
```

- Values are accessed with **dot notation** (`student.name`) or bracket notation (`student["name"]`).
- **this** inside a method refers to the object itself.
- New properties can be added at any time: `student.course = "CSE";`

JavaScript also provides ready-made **built-in objects**: `Math` (`Math.sqrt(16)` gives 4), `String` (text methods like `length` and `toUpperCase()`), `Date` (current date and time) and `Array`. Almost everything in JavaScript — including the document itself in the DOM — is handled as an object, which is why this concept is central to the language.

Document Object Model

The **Document Object Model (DOM)** is the browser's representation of a loaded web page as a **tree of objects**. When a page loads, the browser converts every HTML element,

attribute and piece of text into a node of this tree, with document at the root, then html, head and body, and all elements below them. JavaScript programs work on the page through this tree — the page itself is not edited, the DOM is, and the display updates immediately.

Common DOM methods and properties:

```
document.getElementById("demo")
document.getElementsByClassName("note")
document.querySelector("p")
element.innerHTML = "New text";
element.style.color = "red";
element.value (for form inputs)
```

Example in full: after `<p id="demo">Old</p>`, the statement `document.getElementById("demo").innerHTML = "New text";` replaces Old with New text on the screen. The DOM is the standard (W3C) way for scripts to access and update the content, structure and style of a document.

HTML events and calling JavaScript functions on events

An **event** is something that happens to an element — usually an action by the user. JavaScript code is commonly written as a function and then attached to an event, so it runs exactly when that event occurs.

Common events:

- **onclick** — an element is clicked.
- **onmouseover** / **onmouseout** — the mouse enters or leaves an element.
- **onchange** — the value of a form element changes.
- **onsubmit** — a form is submitted (used for validation before sending).
- **onload** — the page or an image has finished loading.
- **onkeydown** / **onkeyup** — a key is pressed or released.

Calling a function on an event — the standard pattern:

```
function greet() {
  alert("Hello, Student");
}
<button onclick="greet()">Say Hello</button>
```

Here `greet()` is defined once in a script and is called only when the button is clicked. This event-driven style is the normal way interactive pages are built — validate on submit, calculate on click, preview on change.

Likely question: Write a JavaScript program using a function to find the larger of two numbers passed as parameters. Explain the DOM and show how `getElementById` is used to change the content of a paragraph on a button click.

Unit 5 — PHP (3 Periods, 25%)

Topics: Server-side scripting; arrays, functions and forms; advanced PHP; databases — basic commands with PHP examples, connection to server, creating a database, selecting a database, listing databases and table names, creating a table, inserting data, altering tables, queries, deleting a database, deleting data and tables; phpMyAdmin and database bugs.

Server-side scripting

Server-side scripting means the script runs on the **web server**, not in the visitor's browser. The server executes the code, often using a database, and sends only the resulting HTML to the browser. **PHP** is the syllabus language for this; other examples are ASP.NET and server-side JavaScript.

Working, step by step:

- The browser requests a .php page.
- The server passes the file to the PHP engine, which executes the code between `<?php` and `?>` tags; plain HTML outside the tags passes through unchanged.
- PHP may connect to a database, check a login, or calculate values.
- The final output — pure HTML — is sent to the browser. The visitor never sees the PHP code, even in View Source.

Contrast with client-side scripting (JavaScript): runs in the browser, code is visible, good for instant interaction but cannot by itself save data to the server. Server-side code is needed for logins, forms saved to a database, and any content that must stay secure or centralised. PHP needs a server environment such as Apache with PHP and MySQL, commonly installed together as XAMPP or WAMP for practicals.

Arrays, functions and forms in PHP

PHP code is written between `<?php` and `?>` and every statement ends with a semicolon. Variables start with `$` and need no declaration.

Arrays — hold many values in one variable:

```
$fruits = array("Apple", "Mango", "Banana");  
echo $fruits[1]; // Mango (index starts at 0)  
$student = array("name"=>"Aarav", "marks"=>88); // associative array  
echo $student["marks"];
```

Functions — reusable blocks, like in JavaScript:

```
function greet($name) {  
    return "Hello, " . $name;  
}  
echo greet("Student");
```

Forms — an HTML form sends data to a PHP page (action attribute); PHP receives it in ready-made arrays: **\$_POST** for method="post" and **\$_GET** for method="get", indexed by the input's name:

```
<form method="post" action="welcome.php">
<input type="text" name="sname">
<input type="submit">
</form>
```

In welcome.php: echo \$_POST["sname"];

Note: count(\$fruits) gives the number of elements stored in an array, and a loop is normally used to display all its values one by one on the page.

Advanced PHP

In syllabus usage, **advanced PHP** refers to the features needed for complete, real web applications beyond basic syntax. The points most commonly expected are:

- **Working with forms in depth** — validating input on the server (never trust browser checks alone), keeping values after errors, and handling file uploads from forms.
- **Sessions and cookies** — HTTP is stateless, so a **cookie** stores small data in the browser, while a **session** stores user data on the server identified by a session id; sessions are the standard way to keep a user logged in across pages (session_start(), \$_SESSION).
- **File handling** — opening, reading, writing and closing files on the server (fopen, fread, fwrite, fclose).
- **Date and include facilities** — date() for formatted dates, and include/require to reuse common code such as headers and database connections across pages.
- **Error handling and security basics** — checking results of operations, and protecting forms against empty or harmful input before using it in a database query.

For viva, frame the answer honestly: advanced PHP = forms + sessions/cookies + files + database-driven pages working together.

Databases — basic commands with PHP examples, connection to server, creating, selecting and listing databases

PHP works with MySQL using the **mysqli** functions (or PDO). The pattern for every database task is the same: connect → select or create the database → run SQL with mysqli_query() → check the result → close the connection.

Connection to server:

```
$conn = mysqli_connect("localhost", "root", "", "");
if (!$conn) { die("Connection failed"); }
```

The four arguments are server name, username, password and (optionally) database name. On a default XAMPP installation the user is root with an empty password — acceptable only for local practice, never on a real server.

Creating a database:

```
$sql = "CREATE DATABASE school";  
mysqli_query($conn, $sql);
```

Selecting a database (when it was not named in the connection):

```
mysqli_select_db($conn, "school");
```

Listing databases and table names — done by running the SQL SHOW commands and reading the result rows:

```
$res = mysqli_query($conn, "SHOW DATABASES");  
$res = mysqli_query($conn, "SHOW TABLES");  
while ($row = mysqli_fetch_array($res)) { echo $row[0] . "<br>"; }
```

Creating a table, inserting data, altering tables and queries

Once the database is selected, tables are managed with SQL sent through `mysqli_query()`:

Creating a table:

```
$sql = "CREATE TABLE student (  
id INT AUTO_INCREMENT PRIMARY KEY,  
name VARCHAR(50), marks INT)";  
mysqli_query($conn, $sql);
```

Inserting data:

```
$sql = "INSERT INTO student (name, marks)  
VALUES ('Meera', 91)";  
mysqli_query($conn, $sql);
```

Altering a table

 — changing its structure afterwards:

```
$sql = "ALTER TABLE student ADD email VARCHAR(100)";  
mysqli_query($conn, $sql);  
(ALTER can also MODIFY a column's type or DROP a column.)
```

Queries (retrieving data):

```
$res = mysqli_query($conn, "SELECT * FROM student WHERE marks >= 60");  
while ($row = mysqli_fetch_assoc($res)) {  
echo $row["name"] . " - " . $row["marks"] . "<br>";  
}
```

Note the difference in handling: CREATE, INSERT and ALTER return true/false, while SELECT returns a result set that must be read row by row with a fetch function inside a loop. Always check the return value and test on sample data.

Deleting a database, deleting data and tables

PHP performs deletions by sending the matching SQL command through `mysqli_query()` — the PHP side is identical to other commands; only the SQL differs. Learn the three levels clearly, because they are frequently confused:

Deleting data (rows) — the table remains:

```
$sql = "DELETE FROM student WHERE id = 3";
```

```
mysqli_query($conn, $sql);
```

Without a WHERE clause, DELETE removes **all rows** of the table (the empty structure stays).

Deleting a table — structure and data both are removed permanently:

```
$sql = "DROP TABLE student";
```

```
mysqli_query($conn, $sql);
```

Deleting a database — the whole database with all its tables is removed:

```
$sql = "DROP DATABASE school";
```

```
mysqli_query($conn, $sql);
```

- None of these can be undone by PHP; take a backup (export) before running them on real data.
- In practicals, echo a confirmation message only after checking that `mysqli_query()` returned true.
- Close the connection at the end: `mysqli_close($conn);`

phpMyAdmin and database bugs

phpMyAdmin is a free, web-based tool for administering MySQL/MariaDB databases through the browser, installed with XAMPP/WAMP and opened locally at `localhost/phpmyadmin`.

Instead of typing every command, the user clicks through panels:

- Create and drop databases and tables, and design columns with types and keys using forms.
- Browse, insert, edit and delete rows directly in a grid.
- Run any SQL in its SQL tab, and import/export databases (.sql files) for backup and transfer.
- Manage users and privileges.

It is ideal for learning and for checking what your PHP code has actually done in the database — write the code, then verify the result in phpMyAdmin.

Database bugs means the common database faults found while developing: wrong SQL syntax, selecting a database that was never created, table or column name spelling mismatches, duplicate primary-key values, data-type mismatches (text into an INT column), and failed connections due to wrong server, user or password. The practical method of finding them: read the mysqli error (`mysqli_error($conn)`), run the same SQL directly in phpMyAdmin to see the database's own message, correct the cause, and test again with small sample data.

Likely question: Write PHP code to connect to a MySQL server, create a database named college, create a table student(id, name, marks) in it, and insert one record. How is phpMyAdmin useful in this work?

Quick Revision — One Line Each

- The internet is the network; the WWW is one service on it; a browser views pages, a search engine finds them.
- URL = scheme://host:port/path?query#fragment; the fragment is never sent to the server.
- Static pages are fixed files; dynamic pages are built by the server from a database; responsive pages adapt to the screen.
- Every HTML document: DOCTYPE, html, head with title, body with the visible content.
- Selection of tag pairs must never overlap — last opened, first closed is the nesting rule.
- <div> is block-level and is inline; forces a blank space.
- In a form, method post hides data in the request body; the server reads values by the inputs' name attributes.
- CSS priority: inline style is strongest; external CSS in a linked .css file is the recommended method for a whole site.
- Box model, inside out: content, padding, border, margin.
- While loop checks first and may not run; do-while runs at least once.
- DOM is the page as a tree of objects; getElementById is the standard way to reach one element.
- PHP runs on the server and the browser receives only HTML; JavaScript runs in the browser.
- PHP database pattern: connect, run SQL with mysqli_query, fetch SELECT rows in a loop, close.
- DELETE removes rows, DROP TABLE removes the table, DROP DATABASE removes the whole database.

KAMALDNP • kamaldnp.com • support@kamaldnp.com • +91 8318013442 • Varanasi, UP
Prepared by KAMALDNP Academy from the official BTEUP 2026-27 syllabus topics. Short explanations are the author's own plain-English summaries, not syllabus text. Please confirm with your own polytechnic for any local changes.