In Alignment with NEP 2020

WEB TECHNOLOGY

BCA | Semester III | Academic Session 2024 – 2027

Compiled by

Deepak Kumar Tiwari

Asst. Professor, Dept. of BCA RVSCET,
Jamshedpur

Week-wise Index

Quick reference of the 13 weekly modules. Original page numbers refer to the page numbers shown in the footer of each week's content.

TABLE OF CONTENTS (Clickable)

RVS College of Engineering and Technology, Jamshedpur

BCA — 3rd Semester (NEP)

WEB TECHNOLOGY

WEEK 1 — INTERNET FUNDAMENTALS

Course Outcome: CO1 | Topics: Internet basics, WWW, HTTP, Browsers & Servers, Web 2.0

Learning Objectives

By the end of this week, students will be able to:

- Explain the meaning of the Internet and key networking terms.
- Describe what the World Wide Web (WWW) is and how it differs from the Internet.
- Understand how HTTP requests and responses work between a browser and a server.
- Identify the role of web browsers and web servers in delivering web pages.
- Distinguish between Web 1.0, Web 2.0, and Web 3.0 with examples.

1. Internet — Basic Terminologies

1.1 What is the Internet?

The Internet is a worldwide network that connects millions of computers, phones, and other devices together so they can share information. Think of it as a giant highway system where data travels between any two devices anywhere in the world.

It is sometimes called the "network of networks" because it is made up of many smaller networks (homes, offices, colleges, mobile carriers) that are all linked together using common rules called protocols.

> **Quick analogy**
> The Internet is like the postal system: it carries information from one address to another. The roads, post offices, and trucks are the cables, routers, and servers. The letters and parcels are the data packets.

1.2 Important Terms You Must Know

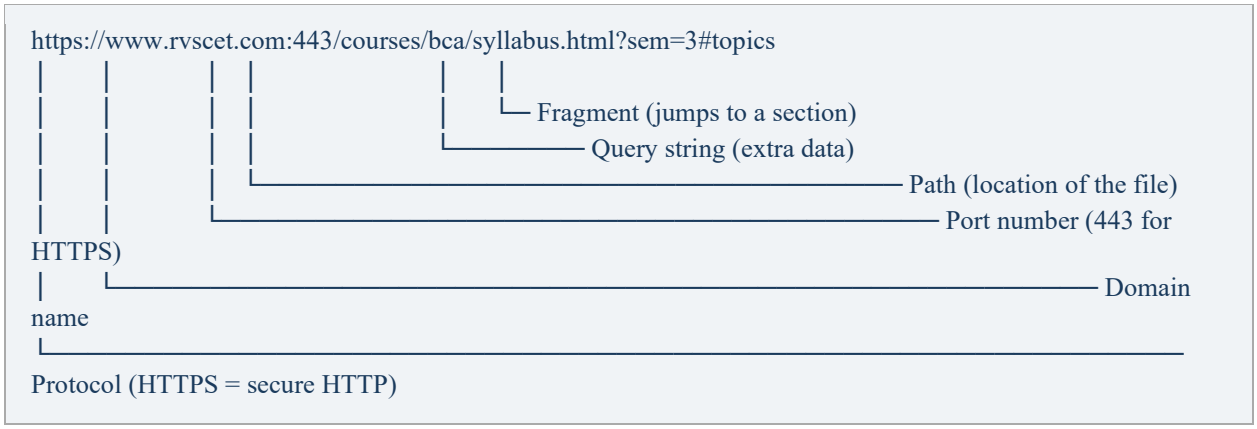
Before going further, let's understand the key terms used throughout the course.

Term	Meaning
Network	Two or more devices connected so they can exchange data. Example: the computers in your college lab connected through cables.

Term	Meaning
Protocol	A set of rules that devices follow to communicate. Example: HTTP, TCP/IP, FTP. Without protocols, computers cannot understand each other.
IP Address	A unique number assigned to every device on the Internet. Example: 142.250.183.46 (one of Google's IPs). It works like a postal address for computers.
Domain Name	A human-friendly name that points to an IP address. Example: www.google.com is easier to remember than 142.250.183.46.
DNS	Domain Name System — the "phonebook" of the Internet. It translates domain names into IP addresses.
URL	Uniform Resource Locator — the full web address of a resource. Example: https://www.rvscet.com/about.html
ISP	Internet Service Provider — the company that gives you Internet access. Example: Jio, Airtel, BSNL.
Bandwidth	The amount of data that can be transferred per second, usually measured in Mbps. Higher bandwidth means faster Internet.
Web Page	A single document written in HTML that opens in a browser.
Website	A collection of related web pages under one domain name.

1.3 Anatomy of a URL

Every web address you type follows a fixed structure. Let's break down a sample URL:



2. World Wide Web (WWW)

2.1 What is the WWW?

The World Wide Web — usually called the Web or simply WWW — is a service that runs on top of the Internet. It is a huge collection of web pages, images, videos, and other documents that are linked together using hyperlinks and accessed through a web browser.

Tim Berners-Lee — the inventor of the Web

The WWW was invented by Sir Tim Berners-Lee in 1989 at CERN, Switzerland.

He created the first web browser, the first web server, and the first website (info.cern.ch) in 1991.

He also defined HTML, HTTP, and URL — the three building blocks of the Web that we still use today.

2.2 Internet vs. WWW (Common Confusion)

Many beginners think "Internet" and "WWW" are the same thing. They are not. The Internet is the infrastructure (the network), while the WWW is just one of many services that use that infrastructure.

Feature	Internet	World Wide Web
Nature	Global network of physical hardware (cables, routers, servers).	Collection of linked documents and resources.
What it carries	All kinds of data (email, file transfer, video calls, gaming).	Web pages, websites, hyperlinks.
Year started	1969 (as ARPANET)	1989–1991 (Tim Berners-Lee)
Main protocol	TCP/IP	HTTP / HTTPS
Access tool	Many- email apps, FTP clients, browsers, etc.	A web browser.
Simple way to think	The roads.	The shops on those roads.

2.3 Components of the WWW

- Web Pages — documents written mostly in HTML.
- Web Browser — software that displays web pages (Chrome, Firefox, Edge).
- Web Server — computer that stores web pages and sends them when requested.
- HTTP / HTTPS — rules used to transfer web pages between server and browser.
- URL — the address used to find a specific page.
- HTML — the language used to write web pages.

2.4 How a Web Page Reaches You — Visual Overview

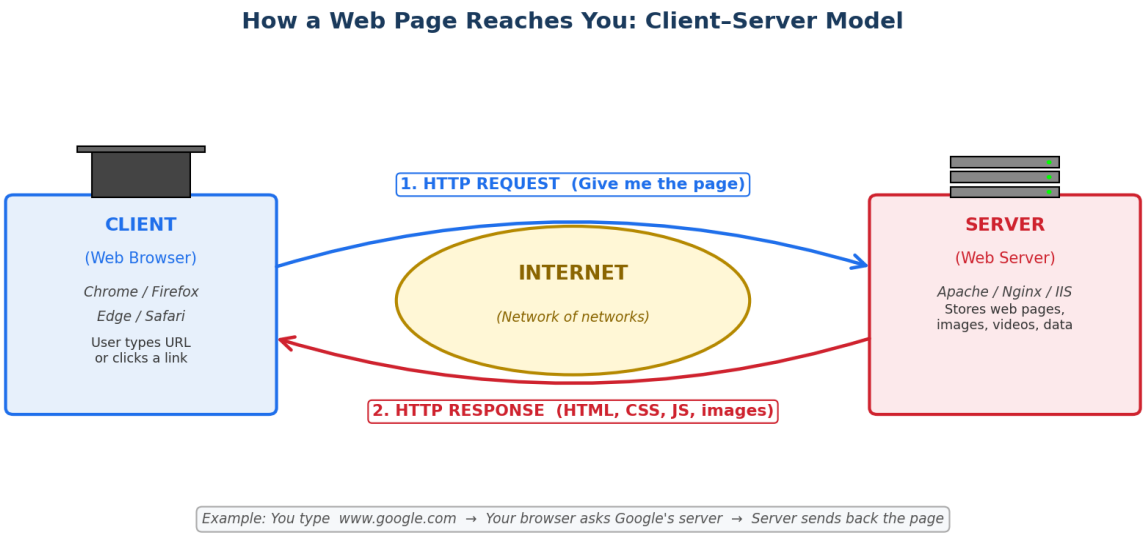


Figure 1.1 — The Client–Server model: how your browser fetches a page from a web server.

3. HTTP — Requests and Responses

3.1 What is HTTP?

HTTP stands for HyperText Transfer Protocol. It is the set of rules that browsers and servers follow when they talk to each other on the Web. Every time you open a website, your browser uses HTTP to ask the server for the page, and the server uses HTTP to send the page back.

HTTPS is simply HTTP with security (encryption added through SSL/TLS), so that no one in between can read what you are sending or receiving — important for passwords, banking, etc.

3.2 HTTP is a Request–Response Protocol

HTTP works in pairs: the client (browser) sends a request, and the server replies with a response. Each request is independent — HTTP is called a stateless protocol because the server does not remember previous requests on its own.

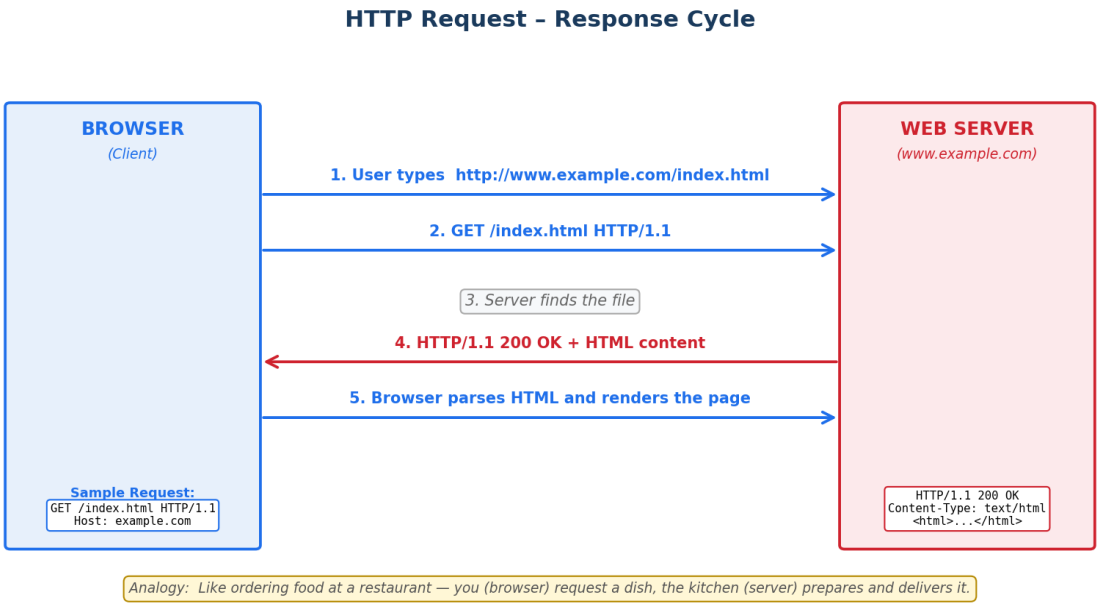


Figure 1.2 — Step-by-step HTTP request–response cycle between a browser and a web server.

3.3 Common HTTP Methods

An HTTP method tells the server what action the client wants to perform.

Term	Meaning
GET	Ask the server for a resource (e.g. open a web page or load an image). Most common method.
POST	Send data to the server (e.g. submitting a registration form, uploading a file).
PUT	Update an existing resource on the server.
DELETE	Delete a resource on the server.
HEAD	Same as GET but the server returns only headers, not the body. Useful for checking if a page exists.

3.4 Structure of an HTTP Request

A typical HTTP request from a browser looks like this:

```
GET /about.html HTTP/1.1
Host: www.rvscet.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Chrome/120)
Accept: text/html,application/xhtml+xml
Accept-Language: en-IN
Connection: keep-alive
```

```
(empty line — request body would go here for POST)
```

- **Request line:** GET /about.html HTTP/1.1 — method, path, and HTTP version.
- **Headers:** extra information like which host, which browser, what content the client can accept.
- **Body:** data sent to the server (used in POST/PUT, empty in GET).

3.5 Structure of an HTTP Response

The server replies with a response that has a similar structure:

```
HTTP/1.1 200 OK
Date: Sat, 25 Apr 2026 10:32:11 GMT
Server: Apache/2.4.58 (Ubuntu)
Content-Type: text/html; charset=UTF-8
Content-Length: 1842

<!DOCTYPE html>
<html><head><title>About Us</title></head>
<body><h1>Welcome to RVSCET</h1>...</body></html>
```

3.6 HTTP Status Codes

The first line of every response contains a 3-digit status code that tells the browser whether the request succeeded or failed.

Code Range	Meaning	Common Examples
1xx	Informational — request received, continuing.	100 Continue
2xx	Success — request was understood and accepted.	200 OK, 201 Created
3xx	Redirection — the resource has moved.	301 Moved Permanently, 302 Found
4xx	Client error — the browser made a bad request.	400 Bad Request, 401 Unauthorized, 404 Not Found
5xx	Server error — the server failed to handle the request.	500 Internal Server Error, 503 Service Unavailable

Memory tip

200 = OK (success).

404 = Not Found (the famous "page does not exist" error).

Code Range	Meaning	Common Examples
500 = the server itself crashed or made a mistake.		

4. Web Browsers and Web Servers

4.1 Web Browser

A web browser is the software you use to view web pages. When you type a URL or click a link, the browser:

- Sends an HTTP request to the correct web server.
- Receives the HTML, CSS, JavaScript, and images in the response.
- Parses the HTML and builds the page in memory (DOM tree).
- Applies the CSS to style the page.
- Runs the JavaScript to make the page interactive.
- Finally, displays (renders) the page on your screen.

Popular Web Browsers

Term	Meaning
Google Chrome	Most widely used browser. Built on the Blink engine. Made by Google.
Mozilla Firefox	Open-source browser developed by the Mozilla Foundation. Uses the Gecko engine.
Microsoft Edge	Default browser on Windows. Built on Chromium (same base as Chrome).
Apple Safari	Default browser on Apple devices. Uses the WebKit engine.
Opera	Lightweight browser known for built-in VPN and ad-blocker.

4.2 Web Server

A web server is a special computer (and the software running on it) whose job is to store websites and send them to browsers when requested. It is always on, always connected to the Internet, and always listening for incoming HTTP requests on port 80 (HTTP) or port 443 (HTTPS).

A web server's main jobs are:

- Receive HTTP requests from browsers (clients) anywhere in the world.
- Locate the requested file (HTML, image, video, CSS, etc.) on its disk.
- If the request is for dynamic content, run server-side code (PHP, Servlet, JSP, Node.js).
- Send back an HTTP response with the result and a status code.
- Log every visit, handle security, and manage many users at the same time.

Popular Web Server Software

Term	Meaning
Apache HTTP Server	Free, open-source, and the most popular web server in the world. Runs on Linux and Windows.
Nginx	Pronounced "Engine-X". Fast and lightweight. Often used for high-traffic websites.
Microsoft IIS	Internet Information Services — Microsoft's web server, runs on Windows.
LiteSpeed	Commercial server known for speed. Compatible with Apache configurations.
Node.js (with Express)	JavaScript-based server, popular for modern web apps and APIs.

4.3 Web Browser vs. Web Server — Quick Comparison

Feature	Web Browser	Web Server
Role	Client — asks for web pages.	Provider — sends web pages.
Runs on	User's device (PC, phone).	Always-on computer in a data centre.
Initiates communication	Yes (sends the request).	No (waits for requests).
Examples	Chrome, Firefox, Edge.	Apache, Nginx, IIS.
Number of users	One user at a time.	Many users at the same time.

5. Web 2.0 Concepts

5.1 What is Web 2.0?

Web 2.0 is the second generation of the World Wide Web. The term was made popular around 2004 by Tim O'Reilly. Unlike the early Web (Web 1.0) where users could only read content, Web 2.0 allows users to read, write, share, and interact.

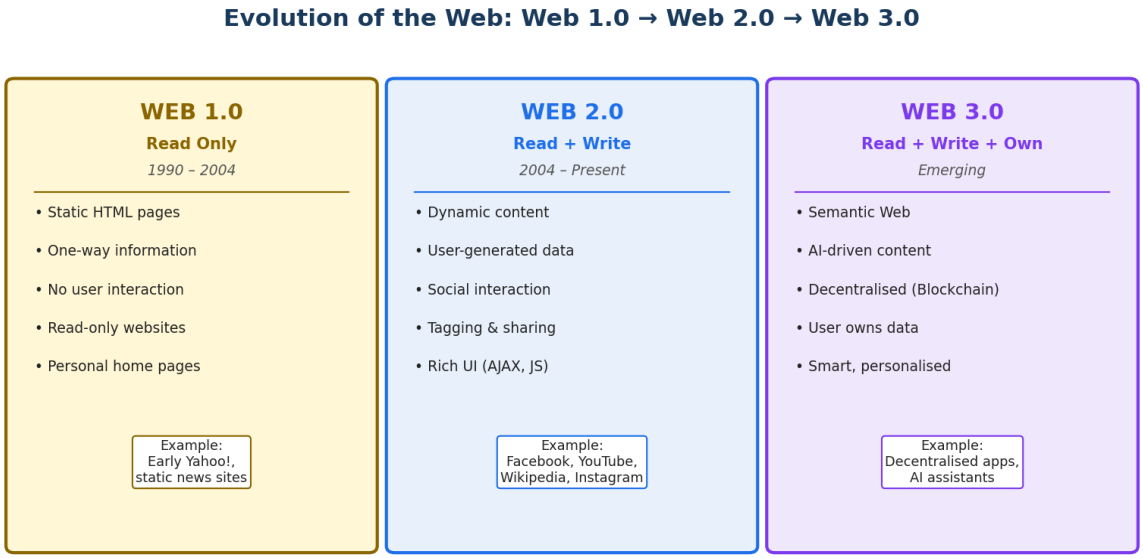


Figure 1.3 — Evolution of the Web from Web 1.0 (read-only) through Web 2.0 (read–write) to Web 3.0 (semantic & decentralised).

5.2 Key Features of Web 2.0

- **User-generated content** — anyone can post, blog, comment, upload videos. (YouTube, Instagram, Wikipedia)
- **Social networking** — people connect, follow, like, and share. (Facebook, X/Twitter, LinkedIn)
- **Rich and interactive interfaces** — AJAX, JavaScript, and frameworks make pages feel like apps.
- **Tagging and folksonomy** — users add their own labels (#hashtags) to organise content.
- **Cloud-based services** — applications run in the browser without installation. (Google Docs, Canva)
- **Mashups** — combining services from different sources. (e.g. Zomato uses Google Maps inside it)
- **RSS feeds and APIs** — websites share their data with other websites and apps.

5.3 Web 2.0 Examples in Daily Life

Term	Meaning
YouTube	You don't just watch videos — you can upload, comment, like, subscribe.
Wikipedia	Anyone in the world can edit articles. Knowledge is built collectively.
Facebook / Instagram	Users create profiles, share posts and stories, react and message.
Google Docs	Multiple people edit the same document at the same time, in the browser.
Stack Overflow	Programmers ask and answer each other's questions and vote on answers.

6. Worked Example — "What happens when you type google.com"

This is one of the most-asked interview questions in web development. Let's trace it step by step using everything we have learned this week.

Stage	What Happens
Step 1	You type www.google.com in the browser address bar and press Enter.
Step 2	The browser checks its cache; if the IP is not found, it asks the DNS server: "What is the IP address of www.google.com?"
Step 3	The DNS server replies with an IP address, for example 142.250.183.46.
Step 4	The browser opens a TCP connection to that IP address on port 443 (HTTPS).
Step 5	The browser sends an HTTP GET request: GET / HTTP/1.1, Host: www.google.com.
Step 6	Google's web server receives the request, finds the home page, and sends back an HTTP response: HTTP/1.1 200 OK with the HTML content.
Step 7	The browser parses the HTML, downloads CSS, JS, and images mentioned inside it (each one a new HTTP request).
Step 8	The browser renders everything on screen, runs the JavaScript, and you see the Google home page.

7. Summary of Week 1

- The Internet is the global hardware network; the WWW is a service of documents that runs on it.
- Every device on the Internet has a unique IP address; DNS turns easy names into IPs.
- HTTP is the protocol used by browsers (clients) and web servers to exchange web pages.
- HTTP works as a request–response cycle, with methods like GET and POST and status codes like 200, 404, 500.
- A web browser is the client (Chrome, Firefox); a web server is the program that stores and serves pages (Apache, Nginx).
- Web 2.0 transformed the Web from read-only to a participatory, social, interactive platform.

8. Practice Questions

A. Short Answer (2 marks each)

1. Define the Internet in your own words.

2. What is the difference between a domain name and an IP address?

3. Expand the abbreviations: HTTP, HTTPS, URL, DNS, ISP, WWW.

4. What is the role of DNS in the Internet?

5. Name any four popular web browsers and any three web servers.

B. Long Answer (5–10 marks each)

1. Explain the client–server model with the help of a neat diagram.

2. Describe the structure of an HTTP request and an HTTP response with examples.

3. Compare Web 1.0, Web 2.0, and Web 3.0 with at least five points of difference.

4. Explain the working of a web browser when a user opens a website.

5. List and explain at least five categories of HTTP status codes with examples.

C. Think and Apply

1. When you open www.youtube.com, identify which parts of the URL represent the protocol, the domain, and the path.

2. Why is HTTP called a stateless protocol? Give a real-life situation where statelessness causes a problem on a website.

3. Wikipedia is often given as the perfect example of Web 2.0. Justify why, using at least three Web 2.0 features.

WEEK 2 — WEB DESIGN BASICS

Course Outcome: CO1 | Topics: Web design, user-centric design, layout, navigation, planning

Learning Objectives

By the end of this week, students will be able to:

- Define web design and distinguish it from web development.
- Apply the principles of user-centric design (UCD) to a website.
- Identify the standard regions of a web page layout and apply layout principles.
- Choose an appropriate navigation pattern and site structure for a project.
- Follow the seven-phase website planning process to plan a real project.

1. Web Design — The Basics

1.1 What is Web Design?

Web design is the process of planning, organizing, and visually arranging the content of a website so that users can find what they want easily and have a pleasant experience using it. It deals with the look, feel, layout, colors, fonts, images, and the overall user experience of a website — everything the visitor sees and interacts with.

Good web design is not just about making a site "look pretty". It is about communicating the right message to the right audience and helping them complete their task — whether that is reading an article, buying a product, or filling a form.

> **Quick analogy**
> Web design is to a website what architecture and interior design are to a building. The web developer is like the contractor who actually constructs it. A beautiful building that no one can find the entrance to is bad architecture — and the same is true of websites.

1.2 Web Design vs. Web Development

Beginners often confuse web design with web development. They are related but different roles.

Aspect	Web Design	Web Development
Focus	How the site looks and feels.	How the site works and behaves.
Main outputs	Wireframes, mockups, color palettes, fonts.	HTML, CSS, JavaScript, server-side code.
Tools used	Figma, Adobe XD, Photoshop, Canva.	VS Code, browsers, Git, Node.js, frameworks.
Skills needed	Visual sense, typography, UX, color theory.	Programming, debugging, databases, deployment.
Question it answers	Does the user enjoy using this site?	Does the site work correctly and quickly?

1.3 Three Dimensions of Web Design

Every web design decision falls under one of three dimensions:

- **Visual design —** colors, typography, images, spacing, alignment, and overall aesthetics.
- **Content design —** the words on the page, headings, calls-to-action, tone of voice, and how the message is structured.
- **Interaction design —** how the user moves through the site: clicks, hovers, animations, forms, and feedback.

A successful website balances all three. A site can have beautiful visuals but fail because the content is confusing, or great content presented in a layout no one wants to read.

2. User-Centric Design (UCD)

2.1 What is User-Centric Design?

User-centric design — also called user-centered design — is a design approach where every decision is made by keeping the end user at the center. Instead of designing whatever the developer or owner thinks is good, the designer studies the actual users, what they want, what frustrates them, and what they are trying to achieve.

The goal of UCD is simple: make the website easy, useful, and pleasant for the people who will actually use it.

2.2 Core Principles of User-Centric Design

Term	Meaning
Know your user	Study the audience: their age, language, technical comfort, devices they use, what they want from the site.
Clarity over cleverness	Use simple, plain language. Don't make the user think. Buttons should clearly say what they do.
Consistency	Use the same colours, fonts, button styles, and layout patterns across the site so users do not get confused.
Feedback	Every action should give a response — a button changes colour on click, a form shows "Saved successfully", an error is highlighted in red.
Accessibility	The site should be usable by everyone, including users with poor vision, slow Internet, or only a keyboard.
Forgiveness	Allow users to undo, go back, or recover easily from mistakes (e.g. "Are you sure you want to delete?").
Speed	Pages should load quickly. Most users leave a site if it takes more than 3 seconds to load.

2.3 The User Persona Idea

To stay focused on real users, designers create user personas — short imagined profiles of typical visitors. Example for a college website:

Sample User Persona — "Aspiring Student Aman"

Age: 18 years. Location: Ranchi. Device: Smartphone (mostly mobile data).

Goals: Find admission dates, see fee structure, check eligibility for BCA.

Frustrations: Slow-loading pages on 4G, broken PDFs, hidden fee details.

What he needs from the site: A clear "Admissions" button on the home page, fast pages, downloadable info, easy contact details.

When the designer is unsure about a design decision, they simply ask: "Will Aman find this useful?" This keeps the website focused on the people it is built for, not on the people building it.

2.4 Why User-Centric Design Matters

- Higher visitor satisfaction and trust.
- Lower bounce rate (users do not leave the site quickly).
- More conversions — registrations, purchases, downloads.
- Less rework, because the design solves real problems from day one.
- Better accessibility, helping the site reach a larger audience.

3. Page Layout Principles

3.1 What is Page Layout?

Page layout is the way different parts of a web page — headings, text, images, menus, ads, links — are arranged on the screen. A good layout guides the user's eye smoothly from the most important information to the least.

3.2 The Five Standard Regions of a Web Page

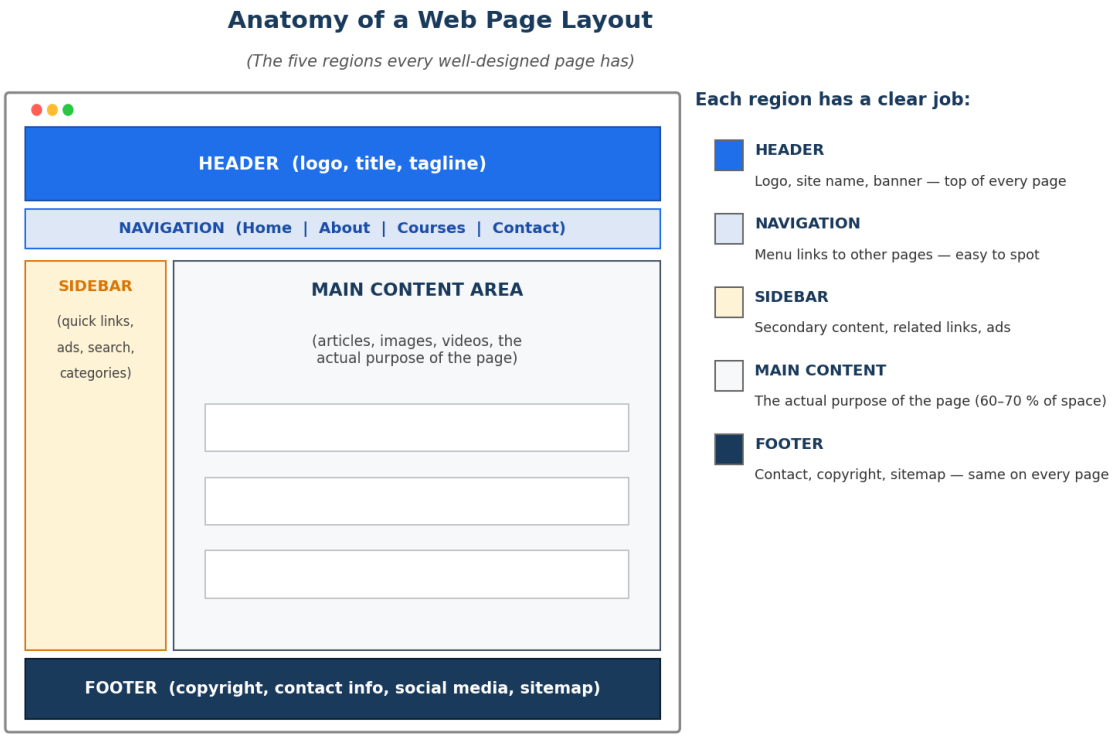


Figure 2.1 — Anatomy of a typical web page layout, showing the five main regions and their roles.

Term	Meaning
Header	The top region. Contains the logo, site name, and sometimes a tagline or banner. Same on every page so users always know where they are.
Navigation	A horizontal or vertical menu with links to the main sections (Home, About, Courses, Contact). The most important navigation aid on the site.

Term	Meaning
Sidebar	A vertical column on the left or right. Contains secondary content like quick links, search box, advertisements, recent posts, or categories.
Main Content	The largest region — usually 60–70 % of the page. This is where the actual purpose of the page lives: the article, product, video, or form.
Footer	The bottom region. Contains copyright, contact information, social media icons, and a sitemap or quick links. Like the header, it is the same on every page.

3.3 Types of Layouts

Layout Type	How it Behaves	Best Used For
Fixed Layout	Width is fixed in pixels (e.g. 960 px). Looks the same on every screen.	Older websites; rarely used today.
Fluid Layout	Width is in percentages. Stretches to fill the screen.	Sites where content reflows naturally (blogs).
Responsive Layout	Adapts to screen size — desktop, tablet, mobile — using CSS media queries.	Almost all modern websites.
Grid Layout	Page divided into rows and columns. CSS Grid or Bootstrap grid is used.	Magazine-style sites, dashboards, e-commerce.
Single-Page Layout	All content on one long scrolling page.	Portfolios, landing pages, product launches.

3.4 Key Layout Principles

- **Visual hierarchy —** the most important item should look the most important. Use size, colour, and position to guide the eye.
- **Balance —** distribute elements so the page does not feel heavy on one side. Can be symmetric (formal) or asymmetric (modern).
- **Alignment —** line up edges of text, buttons, and images. Random placement looks unprofessional.
- **Proximity —** group related items together. Items that belong to the same idea should sit close to each other.
- **White space (negative space) —** the empty area around elements. White space is not wasted space; it makes a page easier to read and feel less cluttered.
- **Contrast —** use light vs. dark, big vs. small, bold vs. thin to make important things stand out.

- **Repetition / consistency —** reuse the same fonts, colours, and button styles to give the site a unified personality.

The F-Pattern and Z-Pattern — how users actually read a page

Eye-tracking studies show that users do not read web pages line by line. On text-heavy pages they scan in an F-shape (horizontally across the top, then horizontally a bit lower, then vertically down the left side).

On visual pages with little text, they follow a Z-shape (top-left → top-right → middle → bottom-right).

Place the most important content (logo, headline, call-to-action) along these natural paths.

4. Navigation and Site Structure

4.1 What is Web Navigation?

Navigation is the system of links and menus that helps users move from one page to another within a website. Good navigation answers three questions for the user at all times: Where am I? Where can I go? How do I get back?

4.2 Common Navigation Patterns

Term	Meaning
Top (horizontal) menu	The most familiar pattern — main links along the top of the page. Easy to scan, works well on desktops.
Side (vertical) menu	A list of links along the left side. Common in dashboards and admin panels where many sections exist.
Hamburger menu	Three short horizontal lines (☰) that open a hidden menu. Saves space on mobile devices.
Breadcrumb	A small text trail showing the path: Home › Courses › BCA › Semester 3. Helps users see where they are inside a deep site.
Mega menu	A large dropdown panel with many links arranged in columns — used by large sites like Amazon and Flipkart.
Footer navigation	A simpler menu repeated at the bottom of the page for users who scrolled all the way down.
Search bar	Not a menu, but the most powerful navigation tool for content-heavy sites — lets users jump anywhere instantly.

4.3 Types of Site Structure

Site structure is the way pages of a website are connected to each other. The structure determines how a user (and a search engine) moves through the site.

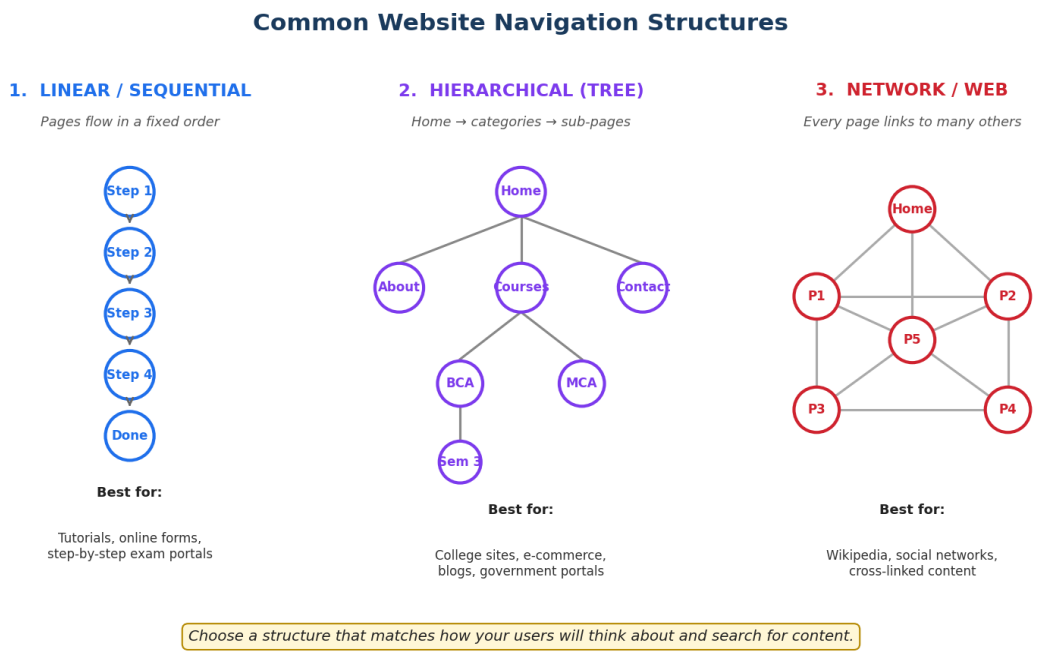


Figure 2.2 — Three common website navigation structures: linear, hierarchical, and network.

Term	Meaning
Linear / Sequential	Pages follow each other in a fixed order, like the pages of a book. The user must complete page 1 before going to page 2. Used in tutorials, online quizzes, and step-by-step forms.
Hierarchical (Tree)	A home page at the top with categories below it, and sub-pages under each category. The most popular structure — used by college sites, e-commerce stores, and blogs.
Network / Web	Every page can link to many other pages. There is no fixed path. Wikipedia is the best example — any article can lead anywhere through hyperlinks.
Hub-and-Spoke	A central page (the hub) connects to several outer pages (the spokes), and the user always returns to the hub. Common in mobile apps and dashboards.

4.4 Best Practices for Navigation

- Keep the main menu short — 5 to 7 items is ideal.
- Use clear, plain words like "Courses" or "Contact", not vague terms like "Stuff" or "Resources".
- Highlight the current page in the menu so the user knows where they are.
- Make the logo clickable — it should always go back to the home page.
- Follow the Three-Click Rule: the user should reach any important page in three clicks or fewer.
- Provide a search bar for content-heavy sites.

- Make navigation reachable on mobile — touch targets at least 44 × 44 pixels.

> **Three-Click Rule**
> A common (though debated) guideline: any important information should be reachable within three clicks from the home page. If users have to click more, they get frustrated and leave. The deeper insight is not the number 3 — it is that every click should clearly bring them closer to their goal.

5. The Website Planning Process

5.1 Why Plan Before You Code?

Jumping directly into HTML and CSS without planning is the most common reason web projects fail. Planning answers important questions early: What is the goal? Who is this for? What pages do we need? How will users move through the site? Without these answers, the team rebuilds the same screens again and again.

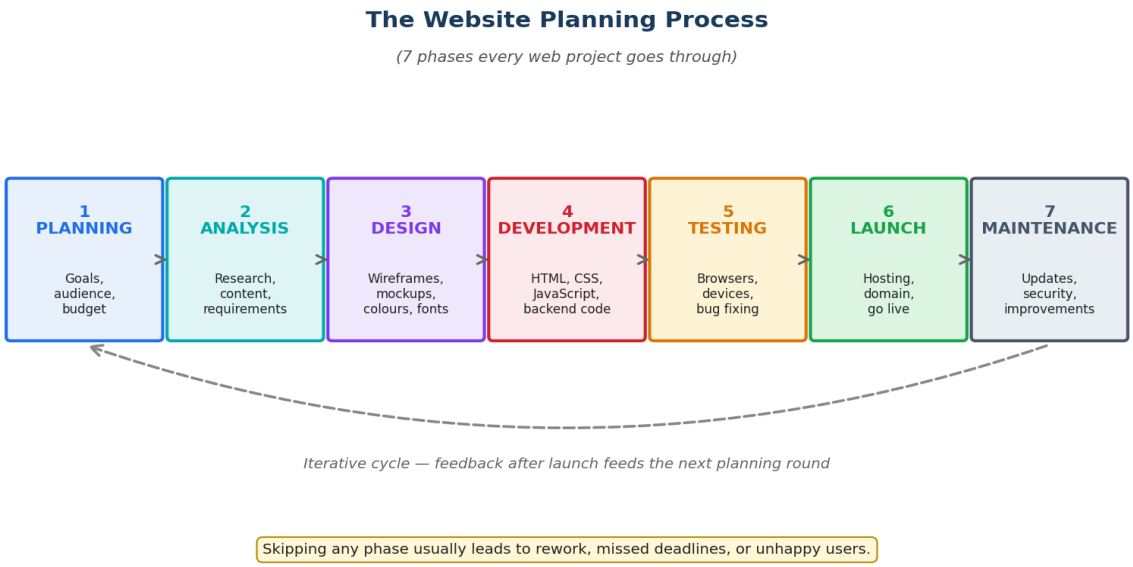


Figure 2.3 — The seven phases of the website planning and development process.

5.2 The Seven Phases Explained

Phase 1 — Planning

Define the purpose of the website (information, sales, registration, branding). Identify the target audience and their needs. Decide the budget, timeline, and team. Prepare a project brief that everyone agrees on.

Phase 2 — Analysis

Gather all detailed requirements. Study competitor websites. Decide on the content needed (text, images, videos, downloads). Prepare a sitemap — a tree showing every page the site will have. Choose the technology stack (HTML/CSS/JS, plus PHP, ASP, or a CMS like WordPress).

Phase 3 — Design

Create wireframes (rough sketches showing the layout of each page). Then create mockups (high-quality designs with real colours, fonts, and images). Decide the colour palette, typography, and image style. Get approval from the client or department before moving on.

Phase 4 — Development

The actual coding phase. Convert the design into working HTML, CSS, and JavaScript. Build server-side functionality (login, database, forms) using PHP, Servlets, JSP, or another backend. Connect the database. Make the site responsive for mobile devices.

Phase 5 — Testing

Check the site on multiple browsers (Chrome, Firefox, Edge), multiple devices (laptop, tablet, mobile), and on different screen sizes. Test every form, every link, every button. Fix bugs. Test for speed, security, and accessibility.

Phase 6 — Launch (Deployment)

Buy a domain name (e.g. www.rvscet.com). Buy hosting space on a server. Upload all files. Configure the database on the server. Test once more on the live URL. Announce the website to users.

Phase 7 — Maintenance

A website is never truly "finished". Update content regularly. Apply security patches. Add new features based on user feedback. Monitor traffic and performance. Backup the database. Plan the next round of improvements — which feeds back into Phase 1.

6. Worked Example — Planning the BCA Department Website

Let's apply everything from this week to a realistic scenario: planning a new website for the BCA Department of RVSCET. We will follow the seven phases briefly.

Phase	Decisions for the BCA Department Website
1. Planning	Goal: Inform aspiring students and showcase the department. Audience: 12th-pass students, parents, current students, recruiters. Budget: low. Timeline: 1 month.
2. Analysis	Pages needed: Home, About, Faculty, Courses, Syllabus, Placements, Events, Contact. Tech: HTML5 + CSS3 + JavaScript + PHP + MySQL.
3. Design	Wireframe each of the 8 pages. Colours: navy + white + a warm accent. Mockups: home page hero with a banner image of the campus, clear "Apply Now" button on the right (Z-pattern).
4. Development	Build HTML pages, style with CSS, add a contact form (PHP), build a faculty list page that pulls from MySQL. Make every page responsive.
5. Testing	Test on Chrome and Firefox, on a laptop and a phone. Check every link. Test the contact form. Fix the slow image on the home page.

Phase	Decisions for the BCA Department Website
6. Launch	Register rvscet-bca.com, purchase hosting, upload files, configure MySQL, go live. Share the link with current students and faculty.
7. Maintenance	Add new event posts every month. Update placement statistics every semester. Apply security patches. Add an alumni section in version 2.

User-centric check for the BCA site

Persona: "Aspiring student Aman". On the home page, can Aman find admission information in one click? Yes — "Admissions" is the second item on the top menu and the home page also has a big "Apply Now" button. The site passes the user-centric test for this persona.

7. Summary of Week 2

- Web design plans how a website looks and feels; web development builds how it works.
- User-centric design puts the real user at the centre of every design choice.
- A standard page layout has five regions: header, navigation, sidebar, main content, and footer.
- Layout principles — hierarchy, balance, alignment, proximity, white space, contrast, consistency — apply to every page.
- Three common site structures are linear, hierarchical, and network; pick the one that matches user behaviour.
- Every successful website follows a 7-phase planning process: Plan → Analyse → Design → Develop → Test → Launch → Maintain.

8. Practice Questions

A. Short Answer (2 marks each)

1. Define web design in your own words.

2. Differentiate between web design and web development with two points each.

3. What is a user persona? Why is it useful?

4. List any five regions of a typical web page layout.

5. Name any three navigation patterns and where each is best used.

B. Long Answer (5–10 marks each)

1. Explain the principles of user-centric design with examples.

2. Discuss the key principles of page layout (hierarchy, balance, alignment, proximity, contrast) with neat sketches.

3. Describe the seven phases of the website planning process. Why is skipping any phase dangerous?

4. Compare linear, hierarchical, and network site structures. Give one real-world example for each.

5. Explain any five best practices for designing website navigation, with reasons.

C. Think and Apply

1. You are designing a website for a small bookstore in Jamshedpur. Identify two user personas, list five pages the site should have, and choose a site structure. Justify your choices.

2. Visit any popular Indian government website (e.g. india.gov.in) and identify three things you would change to make it more user-centric. Explain why.

3. A friend says, "My website looks beautiful, but no one stays on it." Using concepts from this week, list three possible reasons and three fixes.

WEEK 3 — HTML5 STRUCTURE AND ELEMENTS

Course Outcome: CO2 | Topics: Document structure, head & body, text, images, links, tables

Learning Objectives

By the end of this week, students will be able to:

- Write a complete HTML5 document with the correct structure.
- Use the head section to add metadata, title, and external resources.
- Use the body section with semantic HTML5 layout tags.
- Format text using headings, paragraphs, and inline formatting tags.
- Insert images and create different types of hyperlinks.
- Build well-structured tables using <table>, <thead>, <tbody>, and <tfoot>.

1. Introduction to HTML5

1.1 What is HTML5?

HTML stands for HyperText Markup Language. It is the standard language used to create web pages. HTML5 is the fifth and current major version of HTML, finalised in 2014. It is the language every web browser understands by default.

HTML is not a programming language — it is a markup language. We use special words called tags to mark up content ("this is a heading", "this is an image", "this is a link") so that the browser knows how to display it.

1.2 What's New in HTML5 (vs. older HTML)

- Simpler doctype: just <!DOCTYPE html> instead of long DTD declarations.
- New semantic tags: <header>, <nav>, <main>, <article>, <section>, <aside>, <footer>.
- Built-in support for audio and video without plugins (<audio>, <video>).
- New form input types like email, date, number, range, and color.
- Native graphics with <canvas> and <svg>.
- Geolocation, drag-and-drop, web storage, and many more APIs.
- Better support for mobile devices and accessibility.

Key idea — markup is meaning

A good HTML5 page tells the browser not just what to display, but what each thing means. A heading is wrapped in <h1>, a navigation menu in <nav>, the main content in <main>. This helps browsers, search engines, and screen readers understand the page correctly.

2. HTML5 Document Structure

2.1 The Skeleton of Every HTML5 Page

Every HTML5 page begins with a fixed skeleton. This basic template will be the starting point for every page you write this semester.

Example 2.1 — A minimal HTML5 page (save as index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>My First Web Page</title>
</head>
<body>
   <h1>Welcome to RVSCET</h1>
   <p>This is my first HTML5 page.</p>
</body>
</html>
```

▶ ***Output in browser:***

Welcome to RVSCET
This is my first HTML5 page.

2.2 Understanding Each Line

Tag	Purpose
<!DOCTYPE html>	Tells the browser this is an HTML5 document. Must be the very first line.
<html lang="en">	The root tag — every other tag goes inside it. The lang attribute tells the browser the page is in English.
<head>	Container for information about the page (metadata).
<meta charset=...>	Says the page uses UTF-8, so symbols and Indian language characters display correctly.
<meta viewport...>	Makes the page mobile-friendly by adapting to the screen width.
<title>	The page name shown on the browser tab.

Tag	Purpose
<body>	Contains everything the user actually sees on the page.
**<h1>, **	A heading and a paragraph — the visible content of this example.
</html>	Every opening tag must have a matching closing tag with a forward slash.

2.3 HTML5 Document Tree (DOM)

When the browser reads an HTML page, it builds it in memory as a tree of tags called the DOM (Document Object Model). The diagram below shows the family tree of tags inside any HTML5 page.

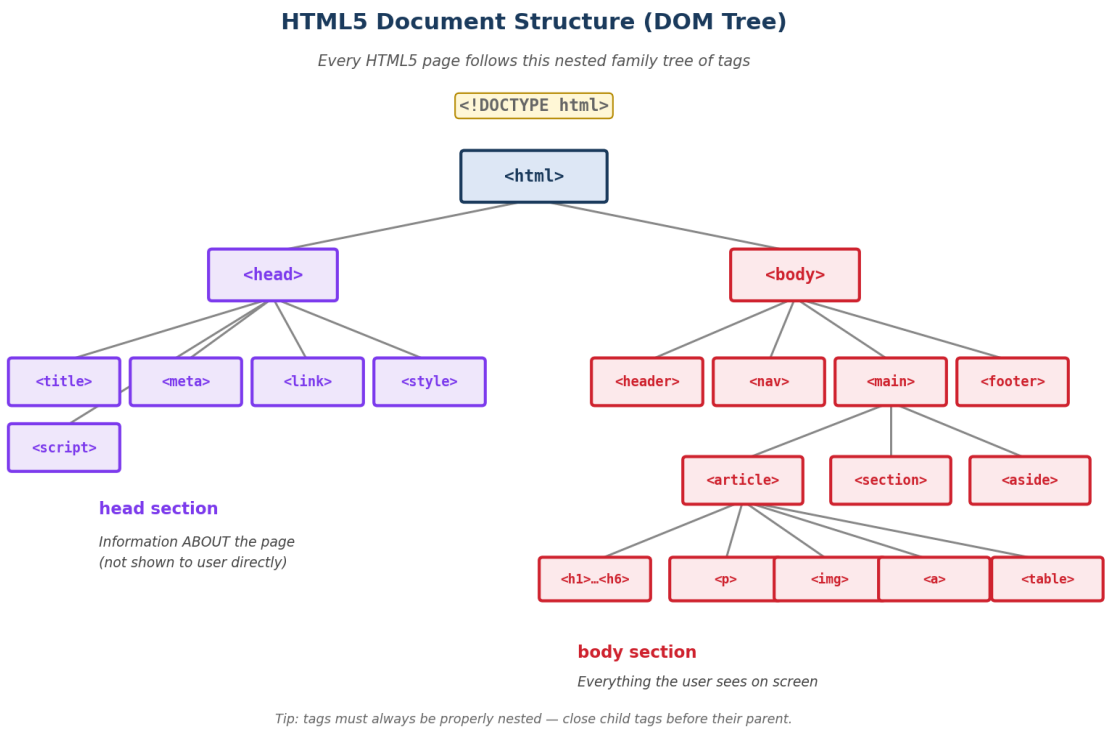


Figure 3.1 — The HTML5 document structure as a DOM tree, showing how tags are nested inside each other.

Golden rules of HTML structure

1. Every page must start with <!DOCTYPE html>.
2. Tags must be properly nested — close child tags before parent tags.
3. Almost every tag has an opening ( ) and a closing ( ) form.
4. Tag names are not case-sensitive but lowercase is the convention.
5. Attributes go inside the opening tag and their values inside double quotes.

3. The Head Section

3.1 What Goes in the Head?

The <head> section contains information ABOUT the page — not what is shown on screen, but instructions for the browser, search engines, and other tools. Think of it as the "settings" of the page.

3.2 Important Tags Inside <head>

Tag	Purpose
<title>	The page title shown on the browser tab and used by search engines (very important for SEO).
<meta>	Metadata: character set, viewport, page description, keywords, author.
<link>	Links external resources to the page — most commonly an external CSS file or favicon.
<style>	Holds CSS rules written directly in the HTML file.
<script>	Holds or links to JavaScript code.
<base>	Sets a base URL that all links on the page use.

3.3 Example — A Realistic Head Section

Example 3.1 — A realistic <head> section with metadata, favicon, CSS link, and inline style

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="BCA Department, RVS College of Engineering and Technology">
  <meta name="keywords" content="BCA, RVSCET, Jamshedpur, computer science">
  <meta name="author" content="Deepak Kumar Tiwary">

  <title>BCA Department | RVSCET Jamshedpur</title>

  <link rel="icon" href="favicon.ico" type="image/x-icon">
  <link rel="stylesheet" href="styles.css">

  <style>
    body { font-family: Arial; }
  </style>
</head>
```

Why metadata matters

The description meta tag often becomes the snippet shown by Google in search results. The keywords help (a little) with ranking. The author tag credits the creator. Together, they make your page discoverable and professional.

4. The Body Section

4.1 What is the Body?

The <body> tag contains the visible part of the web page — everything the user actually sees in the browser window. Headings, paragraphs, images, links, tables, forms, and videos all go inside the body.

4.2 HTML5 Semantic Layout Tags

Before HTML5, developers used to create page sections with , , etc. HTML5 introduced semantic tags that describe what each region IS, not just where it sits. They make code clearer for humans, search engines, and screen readers for visually-impaired users.

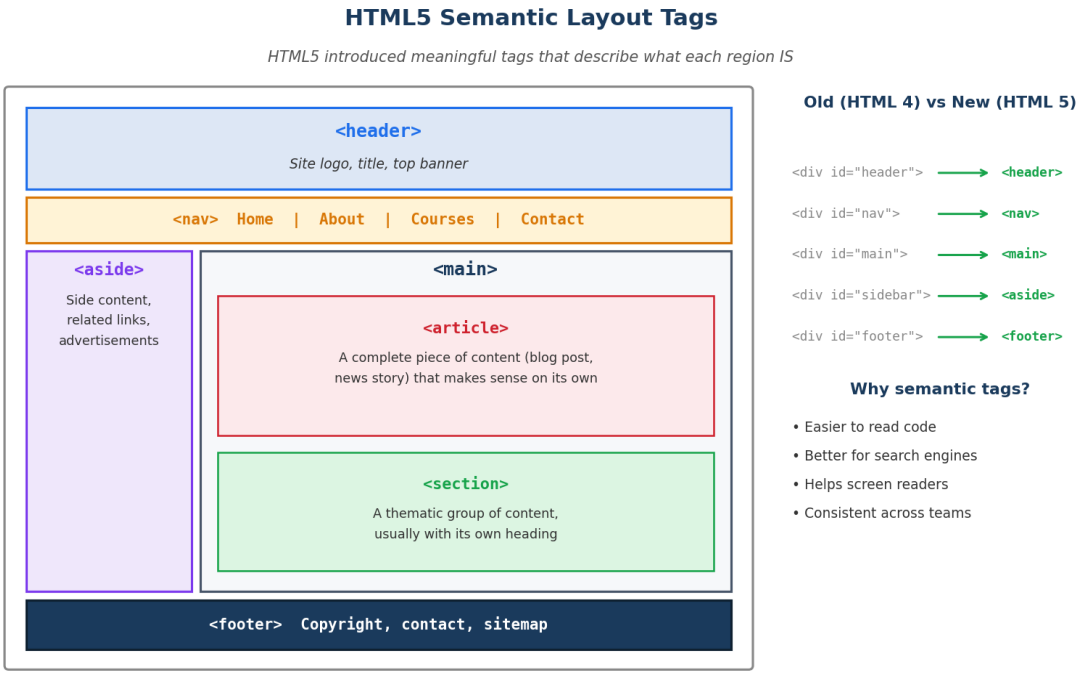


Figure 3.2 — HTML5 semantic layout tags overlaid on a typical web page, with the equivalent old-style code on the right.

Tag	Purpose
<header>	Top region of the page or a section. Contains logo, site title, banner, or section heading.
<nav>	Navigation menu — a group of major links (Home, About, Contact).
<main>	The main content unique to this page. Only one <main> per page.
<article>	Self-contained content that makes sense on its own — a blog post, news story, product card.

Tag	Purpose
<section>	A thematic group of related content, usually with its own heading.
<aside>	Side content related but not essential to the main content — sidebar, ads, related links.
<footer>	Bottom region — copyright, contact info, sitemap. Repeated on every page.
** **	Generic container with no semantic meaning. Use only when no semantic tag fits.
****	Generic inline container — wraps a small piece of text for styling.

4.3 Example — A Page Built with Semantic Tags

Example 4.1 — A semantic HTML5 layout for a library page

```
<body>
  <header>
    <h1>RVSCET Library</h1>
  </header>

  <nav>
    <a href="home.html">Home</a> |
    <a href="books.html">Books</a> |
    <a href="contact.html">Contact</a>
  </nav>

  <main>
    <article>
      <h2>New Arrivals</h2>
      <p>Five new programming books added this week.</p>
    </article>

    <aside>
      <h3>Library Hours</h3>
      <p>Mon-Sat: 9 AM - 6 PM</p>
    </aside>
  </main>

  <footer>
    <p>&copy; 2026 RVSCET, Jamshedpur</p>
  </footer>
</body>
```

5. Text Formatting Elements

5.1 Headings (<h1> to <h6>)

HTML provides six levels of headings, from <h1> (most important) to <h6> (least important). Headings give the page structure and help users (and search engines) scan the content.

Example 5.1 — All six heading levels

```
<h1>Computer Science Department</h1>
<h2>Bachelor of Computer Applications</h2>
<h3>Third Semester Subjects</h3>
<h4>Web Technology</h4>
<h5>Week 3 Topics</h5>
<h6>HTML5 Tags</h6>
```

Heading rules

1. Use only ONE <h1> per page (the main page title).
2. Don't skip levels — go h1 → h2 → h3, not h1 → h4.
3. Use headings for meaning, not for size. Use CSS to control how big they look.

5.2 Paragraphs and Line Breaks

*Example 5.2 — Paragraphs (<p>), line break (
), and horizontal rule (<hr>)*

```
<p>HTML5 is the latest version of the HyperText Markup Language.</p>
<p>It is used by every modern web browser.</p>

<p>This is line one.<br>
This is line two on a new line.</p>

<hr>
<p>This paragraph appears below a horizontal line.</p>
```

Tag	Purpose
<p>	A block of text — a paragraph. Browsers add space before and after.
** **	Inserts a single line break (an empty tag, no closing form). Use sparingly.
<hr>	Horizontal rule — a thematic separator line between sections.
<pre>	Preserves the exact spaces, tabs, and line breaks of the text inside it.

5.3 Inline Text Formatting Tags

These tags style or emphasize a small piece of text inside a paragraph. They do not start a new line.

Example 5.3 — Common inline formatting tags

```
<p>This is <b>bold</b> and this is <strong>strongly important</strong>.</p>
<p>This is <i>italic</i> and this is <em>emphasised</em>.</p>
<p>This text has <u>underline</u> and this is <mark>highlighted</mark>.</p>
```

```
<p>The chemical formula is H<sub>2</sub>O and 2<sup>10</sup> = 1024.</p>
<p>This is <small>small print</small> and this is <del>deleted</del>
   and this is <ins>inserted</ins> text.</p>
<p>Use the <code>&amp;lt;br&amp;gt;</code> tag for a line break.</p>
```

▶ *Output in browser:*

This is bold and this is strongly important.
This is italic and this is emphasised.
This text has underline and this is highlighted.
The chemical formula is H_2O and $2^{10} = 1024$.
This is small print and this is ~~deleted~~ and this is inserted text.
Use the
 tag for a line break.

Tag	Purpose
<b>	Bold text — purely visual.
****	Important text — usually bold, but also tells screen readers it is important.
<i>	Italic text — purely visual.
****	Emphasised text — usually italic, with semantic meaning.
<u>	Underline text.
<mark>	Highlights text (yellow background by default).
<small>	Smaller print, like fine print or copyrights.
<sub>	Subscript — drops below the line (chemistry, footnotes).
<sup>	Superscript — rises above the line (powers, ordinals).
<del>	Deleted text, shown with a strike-through.
<ins>	Inserted text, usually underlined.
<code>	Inline code, shown in a monospace font.

6. Images

6.1 The <img> Tag

The <img> tag inserts an image into a page. Unlike most tags, <img> is empty — it has no closing tag. Instead, the image to be displayed is specified using the src attribute.

Example 6.1 — Basic image tag

```
<img src="campus.jpg" alt="RVSCET campus" width="400" height="250">
```

Tag	Purpose
src	Source — the path or URL of the image file. Required.
alt	Alternative text shown if the image fails to load. Read aloud by screen readers. Required for accessibility.
width	The width of the image in pixels.
height	The height of the image in pixels.
title	Tooltip text shown when the user hovers over the image.
loading	Set to "lazy" to delay loading off-screen images for faster pages.

6.2 Image Path Types

Example 6.2 — Different ways to specify the image path

```
<!-- 1. Same folder as the HTML file -->
<img src="logo.png" alt="Logo">

<!-- 2. Inside a subfolder named 'images' -->
<img src="images/logo.png" alt="Logo">

<!-- 3. From the parent folder (one level up) -->
<img src="../logo.png" alt="Logo">

<!-- 4. Absolute URL — image on another website -->
<img src="https://example.com/images/logo.png" alt="Logo">
```

6.3 Common Image Formats

Format	Best Used For	Notes
.jpg / .jpeg	Photographs and realistic images.	Small file size, supports millions of colours, no transparency.
.png	Logos, icons, screenshots.	Supports transparency, larger files than JPG.
.gif	Simple animations.	Limited to 256 colours.
.svg	Vector graphics — logos, icons.	Scalable to any size without losing quality.
.webp	Modern web format.	Smaller than JPG/PNG with same quality. Supported by all modern browsers.

7. Hyperlinks

7.1 The <a> Tag

Hyperlinks are the heart of the Web — they connect one page to another. The anchor tag <a> creates a link. The href attribute holds the destination URL, and the text between the opening and closing tags is what the user sees and clicks.

Example 7.1 — A basic link

```
<a href="about.html">About Us</a>
```

7.2 Types of Links

(a) External Link — goes to another website

Example 7.2(a) — External link, optionally opening in a new tab

```
<a href="https://www.google.com">Visit Google</a>

<!-- Open in a new browser tab -->
<a href="https://www.google.com" target="_blank">Visit Google in new tab</a>
```

(b) Internal Link — goes to another page on the same site

Example 7.2(b) — Internal link to another HTML file

```
<a href="contact.html">Contact Us</a>
<a href="courses/bca.html">BCA Programme</a>
```

(c) Bookmark Link — jumps to a section of the same page

Example 7.2(c) — Bookmark link inside the same page

```
<!-- Step 1: place an id on the target heading -->
<h2 id="admissions">Admissions Procedure</h2>

<!-- Step 2: link to that id with a # symbol -->
<a href="#admissions">Jump to Admissions</a>
```

(d) Email Link — opens the user's email app

Example 7.2(d) — Email link using mailto:

```
<a href="mailto:bca@rvscet.com">Email the BCA Department</a>

<!-- With subject and body filled in -->
<a href="mailto:bca@rvscet.com?subject=Admission&body=Hello sir,">
   Email about Admission
</a>
```

(e) Phone Link — works on mobile

Example 7.2(e) — Tap-to-call link

```
<a href="tel:+919876543210">Call us: +91 98765 43210</a>
```

(f) Download Link

Example 7.2(f) — Force download instead of opening

```
<a href="syllabus.pdf" download>Download Syllabus (PDF)</a>
```

7.3 Image as a Link

You can wrap an <img> inside an <a> tag to make the image clickable.

Example 7.3 — A clickable logo that links to the home page

```
<a href="home.html">
   <img src="logo.png" alt="RVSCET logo — go to home" width="120">
</a>
```

Useful target values

target="_self" — open in the same tab (default).

target="_blank" — open in a new tab.

target="_parent" / "_top" — used with frames (rare today).

8. Tables in HTML5

8.1 When to Use a Table

HTML tables are designed to display data that has a row-and-column relationship — like a marksheet, a class timetable, or a price list. Tables should NOT be used for page layout (use CSS Grid or Flexbox for that).

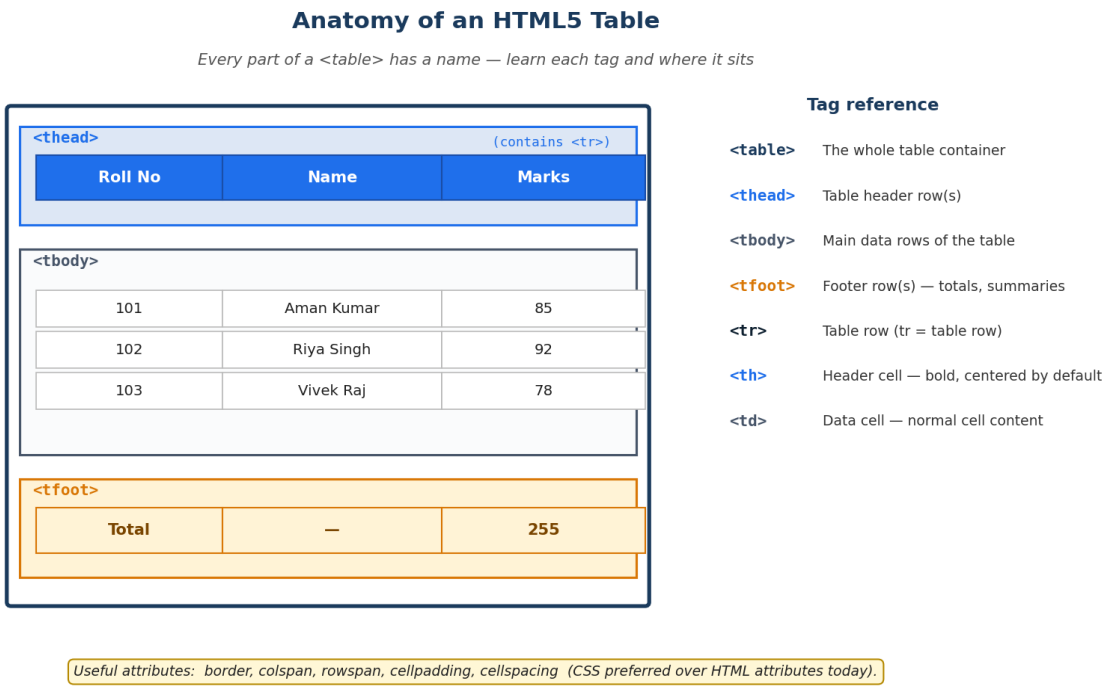


Figure 3.3 — Anatomy of an HTML5 table, showing every standard tag and its location.

8.2 Basic Table Structure

Example 8.1 — A complete student-marks table with thead, tbody, and tfoot

```
<table border="1">
  <thead>
    <tr>
      <th>Roll No</th>
      <th>Name</th>
      <th>Marks</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>101</td>
      <td>Aman Kumar</td>
      <td>85</td>
    </tr>
    <tr>
      <td>102</td>
      <td>Riya Singh</td>
      <td>92</td>
    </tr>
    <tr>
      <td>103</td>
      <td>Vivek Raj</td>
      <td>78</td>
```

```
    </tr>
  </tbody>
  <tfoot>
    <tr>
      <td>Total</td>
      <td>—</td>
      <td>255</td>
    </tr>
  </tfoot>
</table>
```

8.3 Table Tags Explained

Tag	Purpose
<table>	The wrapper for the entire table.
<caption>	An optional title for the table — placed just inside <table>.
<thead>	Group of header rows. Usually contains one row of <th> cells.
<tbody>	Group of data rows — the body of the table.
<tfoot>	Group of footer rows — totals, summaries, or notes.
<tr>	Table row. Stands for "table row".
<th>	Header cell. Bold and centered by default.
<td>	Standard data cell. Stands for "table data".

8.4 Spanning Cells — colspan and rowspan

Sometimes a cell needs to span across multiple columns or rows. Use colspan to span columns and rowspan to span rows.

Example 8.2 — Using colspan and rowspan

```
<table border="1">
  <tr>
    <th colspan="3">Class Timetable — Monday</th>
  </tr>
  <tr>
    <th>Period</th>
    <th>Subject</th>
    <th>Faculty</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Web Technology</td>
```

```
      <td rowspan="2">Mr. Deepak Tiwary</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Web Tech Lab</td>
      <!-- faculty cell continues from above -->
    </tr>
</table>
```

8.5 Useful Table Attributes

Tag	Purpose
border	Width of the table border in pixels (HTML attribute, but CSS is preferred).
cellpadding	Space inside each cell, between the content and the border.
cellspacing	Space between adjacent cells.
colspan="n"	Makes a cell span n columns.
rowspan="n"	Makes a cell span n rows.
align	Horizontal alignment of cell content (left, center, right).

Modern best practice

The HTML attributes border, cellpadding, cellspacing, and align still work, but professionals style tables using CSS — it gives more control and keeps HTML clean. We will study CSS table styling in Week 5.

9. Worked Example — A Complete HTML5 Page

Let's combine everything from this week into one complete page — a simple BCA Department home page using semantic structure, headings, paragraphs, an image, links, and a table.

Example 9.1 — A complete BCA Department home page using all of Week 3's tags

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="BCA Department, RVSCET Jamshedpur">
  <title>BCA Department | RVSCET</title>
</head>
<body>
```

```
  <header>
    <h1>RVSCET — BCA Department</h1>
    <p><em>Empowering tomorrow's developers, today.</em></p>
  </header>

  <nav>
    <a href="index.html">Home</a> |
    <a href="about.html">About</a> |
    <a href="#faculty">Faculty</a> |
    <a href="#timetable">Timetable</a> |
    <a href="mailto:bca@rvscet.com">Contact</a>
  </nav>

  <main>
    <article>
      <h2>Welcome to BCA</h2>
      <img src="campus.jpg" alt="RVSCET campus" width="500">
      <p>The Bachelor of Computer Applications programme at
      <strong>RVSCET, Jamshedpur</strong> is a three-year
      degree designed to prepare students for the IT industry.</p>
    </article>

    <section id="faculty">
      <h2>Our Faculty</h2>
      <p>Dedicated teachers with industry experience.</p>
    </section>

    <section id="timetable">
      <h2>Today's Schedule</h2>
      <table border="1">
        <thead>
          <tr><th>Period</th><th>Subject</th><th>Faculty</th></tr>
        </thead>
        <tbody>
          <tr><td>1</td><td>Web Technology</td><td>Mr. D. K. Tiwary</td></tr>
          <tr><td>2</td><td>Data Structures</td><td>Ms. R. Sinha</td></tr>
          <tr><td>3</td><td>Mathematics</td><td>Dr. P. Verma</td></tr>
        </tbody>
      </table>
    </section>
  </main>

  <footer>
    <p>&copy; 2026 RVSCET, Jamshedpur. All rights reserved.</p>
  </footer>

</body>
</html>
```

How to run this on your computer

1. Open Notepad (or VS Code).
2. Paste the code above.
3. Save the file as bca.html (make sure it does not save as bca.html.txt).
4. Double-click the file — it will open in your default web browser.
5. To edit, right-click → Open with → Notepad / VS Code.

10. Summary of Week 3

- Every HTML5 page starts with <!DOCTYPE html> and is wrapped in <html>.
- Inside <html>, we have <head> (information about the page) and <body> (visible content).
- The <head> holds <title>, <meta>, <link>, <style>, and <script> tags.
- The <body> uses semantic tags — <header>, <nav>, <main>, <article>, <section>, <aside>, <footer> — to give meaning to each region.
- Headings (<h1>–<h6>) and paragraphs (<p>) form the basic text structure.
- Inline tags like , , <mark>, <sub>, <sup> add formatting and emphasis.
- Images are added with <img> using src and alt; supported formats include JPG, PNG, GIF, SVG, WebP.
- Hyperlinks use the <a> tag — for external sites, internal pages, bookmarks, email, phone, and downloads.
- Tables are built with <table>, <thead>, <tbody>, <tfoot>, <tr>, <th>, and <td>; spanning is done with colspan and rowspan.

11. Practice Questions

A. Short Answer (2 marks each)

1. What is the purpose of <!DOCTYPE html>? Where must it appear?

2. Differentiate between <head> and <body> with two points each.

3. Differentiate between <b> and ; <i> and .

4. What is the difference between <h1> and <h6>?

5. Write the HTML to insert an image named photo.jpg with an alt text of "Class photo".

6. Write the HTML for a link that opens www.rvscet.com in a new browser tab.

7. What is the use of colspan and rowspan in a table?

B. Long Answer (5–10 marks each)

1. Explain the structure of an HTML5 document with a neat DOM tree diagram.

2. List and explain at least seven HTML5 semantic layout tags with examples.

3. Describe any eight inline text formatting tags with one-line code examples for each.

4. Explain different types of hyperlinks (external, internal, bookmark, email, phone) with code examples.

5. Describe the structure of an HTML5 table using <thead>, <tbody>, and <tfoot>. Write code for a student marks table with at least three rows and a totals row.

C. Lab / Hands-On Tasks

1. Write an HTML5 page about yourself with a heading, your photo, three paragraphs about your hobbies, and a link to your college's website.

2. Create a class timetable for any one day of the week using a table with <thead>, <tbody>, and at least one cell using rowspan or colspan.

3. Create a single-page personal portfolio that uses every semantic tag (<header>, <nav>, <main>, <article>, <section>, <aside>, <footer>) at least once.

4. Write an HTML page with a Table of Contents at the top using bookmark links that jump to four sections (Introduction, Skills, Projects, Contact) lower on the same page.

WEEK 4 — HTML5 FORMS AND MEDIA

Course Outcome: CO2 | Topics: Form elements, input validation, audio & video, new HTML5 controls

Learning Objectives

By the end of this week, students will be able to:

- Create HTML5 forms using <form>, <input>, <select>, <textarea>, and <button>.
- Use the new HTML5 input types — email, number, date, color, range, and others.
- Apply built-in HTML5 validation using required, pattern, min, max, and maxlength.
- Group and label form fields properly using <label>, <fieldset>, and <legend>.
- Embed audio and video on a web page using <audio>, <video>, and <source>.
- Build a complete registration form with validation and a media-rich page.

1. Introduction to HTML5 Forms

1.1 Why Forms?

A form is the main way a website collects data from users — registration, login, contact, feedback, search, payment, and so on. Without forms there would be no two-way communication on the Web. HTML5 made forms much more powerful: new input types, built-in validation, and better mobile support all without any JavaScript.

Real-life examples of forms

- Gmail login screen — two text inputs (email, password) and a submit button.
- Amazon search bar — a single text input plus a search button.
- Government exam registration — names, dates, photos, payment, and digital signatures.
- College admission portal — personal details, education, document uploads, captcha.
- Online polls and surveys — radios, checkboxes, dropdowns, and free-text questions.

1.2 How Forms Work — The Big Picture

When a user fills a form and clicks Submit, the browser packs all the data and sends it to a server program (usually written in PHP, Servlet, JSP, Node.js, or Python). The server processes the data — saves it to a database, sends an email, or whatever is needed — and sends back a response page.

We will build the front-end (HTML form) this week. The back-end (PHP server) is covered later in Week 10. Until then, we will use a simple action like "register.php" as a placeholder — the form will appear in the browser even without a real server.

2. The <form> Element

2.1 Basic Syntax

Every form is wrapped inside a <form> tag. Two attributes control where and how the data is sent:

Example 2.1 — A minimal form skeleton

```
<form action="register.php" method="POST">
  <!-- form controls go here -->
  <input type="text" name="username">
  <button type="submit">Send</button>
</form>
```

Tag / Attribute	Purpose
action	URL of the server program that will receive and process the form data.
method	How to send the data — usually GET or POST.
name	Required on every input — the server uses this name to read the value.
id	Used by CSS and JavaScript; also used by <label for=...>.
target	Where to display the response: _self, _blank, _parent.
enctype	Encoding for the data — only matters for file uploads (multipart/form-data).
novalidate	Disables HTML5's automatic validation.

2.2 GET vs. POST — Which One Should You Use?

Aspect	GET method	POST method
Where data goes	Appended to the URL after a ? sign.	Sent in the body of the request, hidden from URL.
Visible to user?	Yes — visible in the address bar.	No — not shown in the address bar.
Bookmark-able?	Yes — the URL contains the data.	No — body data is not in the URL.

Aspect	GET method	POST method
Data size	Limited (~ 2,000 characters).	Practically unlimited.
Security	Insecure for sensitive data.	Better for sensitive data (still use HTTPS).
Best for	Search queries, filters, page links.	Login, registration, payments, file uploads.

Rule of thumb

Use GET when the form just retrieves information (like a search). Use POST when the form changes something on the server (registration, posting a comment, placing an order). For any password or personal data, always use POST and serve the page over HTTPS.

3. Anatomy of a Form

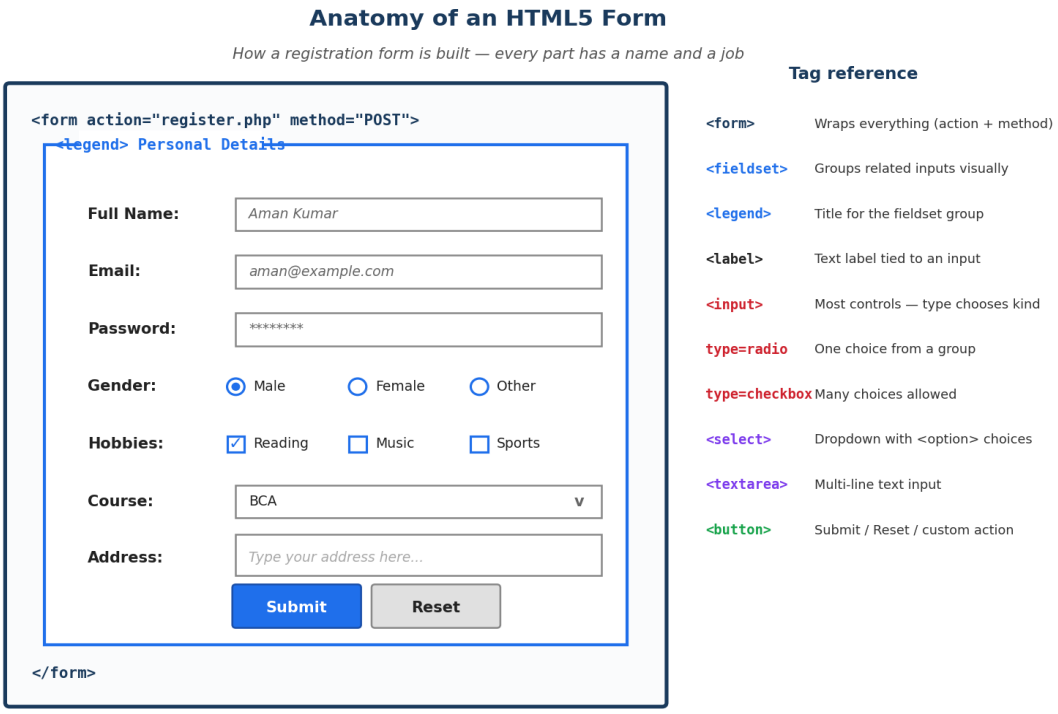


Figure 4.1 — A registration form with every part labelled, plus a tag reference on the right.

4. Common Form Controls

4.1 Single-Line Text Input

Example 4.1 — Text input with a linked label and placeholder

```
<label for="uname">Username:</label>
<input type="text" id="uname" name="username" placeholder="Enter your name">
```

> ***Output in browser:***
>
> Username: [Enter your name]

Tag / Attribute	Purpose
placeholder	Light grey hint text shown inside an empty input.
maxlength	Maximum number of characters the user can type.
readonly	User can read but not edit the value.
disabled	Greys out the field; not sent to the server.
value	The default value when the page loads.

4.2 Password Input

Example 4.2 — Password input that hides characters as user types

```
<label for="pwd">Password:</label>
<input type="password" id="pwd" name="password" minlength="8">
```

4.3 Radio Buttons (choose ONE)

All radios in a group must share the same name attribute. Only one can be selected at a time.

Example 4.3 — Radio buttons for gender (only one selected)

```
<p>Gender:</p>
<input type="radio" id="male"   name="gender" value="M" checked>
<label for="male">Male</label>

<input type="radio" id="female" name="gender" value="F">
<label for="female">Female</label>

<input type="radio" id="other"  name="gender" value="O">
<label for="other">Other</label>
```

> ***Output in browser:***
>
> Gender: (•) Male ( ) Female ( ) Other

4.4 Checkboxes (choose MANY)

Example 4.4 — Checkboxes for hobbies (many selectable)

```
<p>Hobbies:</p>
```

```
<input type="checkbox" id="h1" name="hobbies" value="reading" checked>
<label for="h1">Reading</label>

<input type="checkbox" id="h2" name="hobbies" value="music">
<label for="h2">Music</label>

<input type="checkbox" id="h3" name="hobbies" value="sports">
<label for="h3">Sports</label>
```

> ***Output in browser:***

Hobbies: [✓] Reading [ ] Music [ ] Sports

Radio vs. Checkbox

Radio = one choice from a group (Yes / No, Male / Female).

Checkbox = zero or more choices (select all hobbies that apply).

4.5 Dropdown — <select> with <option>

Example 4.5 — Dropdown menu with a default and pre-selected option

```
<label for="course">Course:</label>
<select id="course" name="course">
  <option value="">-- Select your course --</option>
  <option value="bca" selected>BCA</option>
  <option value="bba">BBA</option>
  <option value="btech">B.Tech</option>
</select>
```

Grouping options with <optgroup>

Example 4.6 — Grouped dropdown using <optgroup>

```
<select name="semester">
  <optgroup label="BCA">
    <option>1st Sem</option>
    <option>2nd Sem</option>
    <option>3rd Sem</option>
  </optgroup>
  <optgroup label="MCA">
    <option>1st Sem</option>
    <option>2nd Sem</option>
  </optgroup>
</select>
```

4.6 Multi-Line Text — <textarea>

Example 4.7 — A 4-row, 40-column text area

```
<label for="addr">Address:</label>
<textarea id="addr" name="address"
        rows="4" cols="40"
        placeholder="Street, City, PIN"></textarea>
```

Tag / Attribute	Purpose
rows	Number of visible text lines.
cols	Visible width in characters.
maxlength	Maximum number of characters allowed.
wrap	soft (default) or hard — controls how the text wraps.

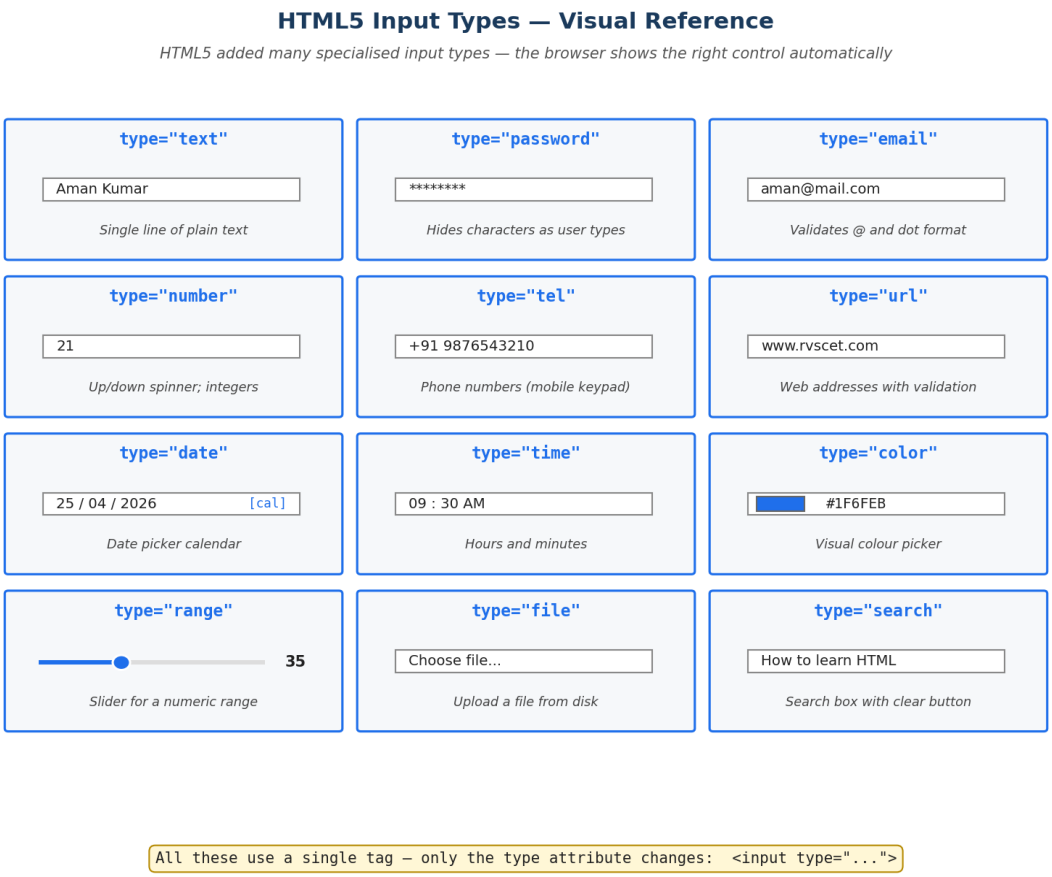
4.7 Buttons

Example 4.8 — Three button types: submit, reset, and button

```
<!-- 1. Submit button (sends the form) -->
<button type="submit">Register</button>
<input type="submit" value="Register">

<!-- 2. Reset button (clears the form) -->
<button type="reset">Clear</button>
<input type="reset" value="Clear">

<!-- 3. Generic button (used with JavaScript) -->
<button type="button" onclick="alert('Hi!')">Say Hi</button>
```

5. New HTML5 Input Types

Older HTML had only a few input types — text, password, radio, checkbox, button, submit. HTML5 added many specialised types. They give the user a better experience and validate themselves automatically.



5.1 Text-Style Inputs

Example 5.1 — Specialised text inputs

```
<input type="email"  name="email"  placeholder="name@site.com">
<input type="url"    name="site"   placeholder="https://...">
<input type="tel"    name="phone"  placeholder="+91 9876543210">
<input type="search" name="q"      placeholder="Search...">
<input type="password" name="pwd">
```

Tag / Attribute	Purpose
type="email"	Validates that the user enters a valid email format. On phones, shows the @-key keypad.
type="url"	Validates web address format (must start with http:// or https://).
type="tel"	Phone number — shows numeric keypad on mobile (does not validate format).
type="search"	Search box — many browsers add a tiny X to clear the value.
type="password"	Hides characters as the user types.

5.2 Numeric Inputs

Example 5.2 — Numeric inputs: spinner and slider

```
<!-- Spinner: integers between 0 and 100 -->
<input type="number" name="marks" min="0" max="100" step="1" value="75">

<!-- Slider: same range, drag to choose -->
<input type="range"  name="score" min="0" max="100" step="5" value="50">
```

Tag / Attribute	Purpose
min	Smallest value allowed.
max	Largest value allowed.
step	Increment per click (1 means whole numbers, 0.1 means decimals).
value	Default value at page load.

5.3 Date and Time Inputs

Example 5.3 — Five date/time inputs (each shows its own picker)

```
<input type="date"           name="dob">
<input type="time"           name="alarm">
<input type="datetime-local" name="appointment">
<input type="month"          name="billing">
<input type="week"           name="week">
```

5.4 Special Inputs

Example 5.4 — color, file, and hidden inputs

```
<!-- Colour picker — opens the system colour palette -->
<input type="color" name="theme" value="#1F6FEB">

<!-- File upload — single or multiple, can restrict types -->
<input type="file" name="photo" accept="image/*">
<input type="file" name="docs"  multiple accept=".pdf,.doc,.docx">

<!-- Hidden field (sent to server but invisible to user) -->
<input type="hidden" name="user_id" value="12345">
```

6. Input Validation

6.1 Why Validation?

Validation means making sure the data the user enters is correct and complete before it is sent to the server. Without validation, a registration form could accept blank names, invalid emails, or negative ages — leading to bad data and bugs.

HTML5 gives you simple, built-in validation through attributes — no JavaScript needed for most basic cases. The browser shows a friendly error message in the user's language.

6.2 Common Validation Attributes

Tag / Attribute	Purpose
required	The field cannot be left empty when submitting.
minlength="n"	Minimum number of characters.
maxlength="n"	Maximum number of characters.
min / max	Smallest and largest values for number, range, and date inputs.
pattern	A regular expression the input must match. Powerful but advanced.
title	Tooltip shown if validation fails (also shown on hover).
novalidate	Set on the <form> tag to disable HTML5 validation.

6.3 Examples

Example 6.1 — Validation in action

```
<!-- Required, minimum 3 characters -->
<input type="text" name="username" required minlength="3">

<!-- Email format checked automatically -->
<input type="email" name="email" required>

<!-- Age must be between 5 and 99 -->
<input type="number" name="age" min="5" max="99" required>

<!-- 10-digit Indian mobile number using pattern -->
<input type="tel" name="mobile"
       pattern="[6-9][0-9]{9}"
       title="Enter a 10-digit mobile number starting with 6, 7, 8, or 9"
       required>

<!-- Strong password: 8+ chars, must have a letter and a digit -->
<input type="password" name="pwd"
       pattern="(?=.*[A-Za-z])(?=.*[0-9]).{8,}"
       title="At least 8 characters with a letter and a digit"
       required>
```

Important
HTML5 validation runs in the browser. A clever user can disable it. Therefore, ALWAYS validate the data again on the server (in PHP, JSP, etc.) before saving it. Browser validation is for user convenience; server validation is for security.

7. HTML5 Audio Element

7.1 Adding Audio to a Page

Before HTML5, audio could only be played using browser plug-ins like Flash. HTML5 introduced the <audio> tag — native audio playback that works in every modern browser without any plug-in.

Example 7.1 — Two ways to add audio

```
<!-- Simple audio player with default controls -->
<audio src="song.mp3" controls></audio>

<!-- Multiple sources for cross-browser support -->
<audio controls autoplay loop>
   <source src="song.mp3" type="audio/mpeg">
   <source src="song.ogg" type="audio/ogg">
   <source src="song.wav" type="audio/wav">
   Your browser does not support the audio element.
</audio>
```

7.2 Audio Attributes

Tag / Attribute	Purpose
controls	Show the built-in player (play/pause, volume, progress).
autoplay	Start playing as soon as the page loads (most browsers block this with sound).
loop	Replay the audio forever.
muted	Start with sound off.
preload	auto / metadata / none — when the browser should load the file.
src	Path or URL of the audio file (alternative to <source> tags).

7.3 Supported Audio Formats

Format	File Extension	Browser Support
MP3	.mp3	All modern browsers — most common choice.
OGG Vorbis	.ogg	Chrome, Firefox, Edge — open-source format.
WAV	.wav	All browsers — large file size.
AAC	.m4a	Safari, Chrome, Edge.

8. HTML5 Video Element

8.1 Adding Video to a Page

The <video> tag works exactly like <audio>, with a few extra attributes for size and a poster image (the picture shown before the video plays).

Example 8.1 — A complete <video> with poster image and three formats

```
<video width="500" height="300" controls poster="thumb.jpg">
   <source src="intro.mp4"  type="video/mp4">
   <source src="intro.webm" type="video/webm">
   <source src="intro.ogg"  type="video/ogg">
   Your browser does not support the video element.
</video>
```

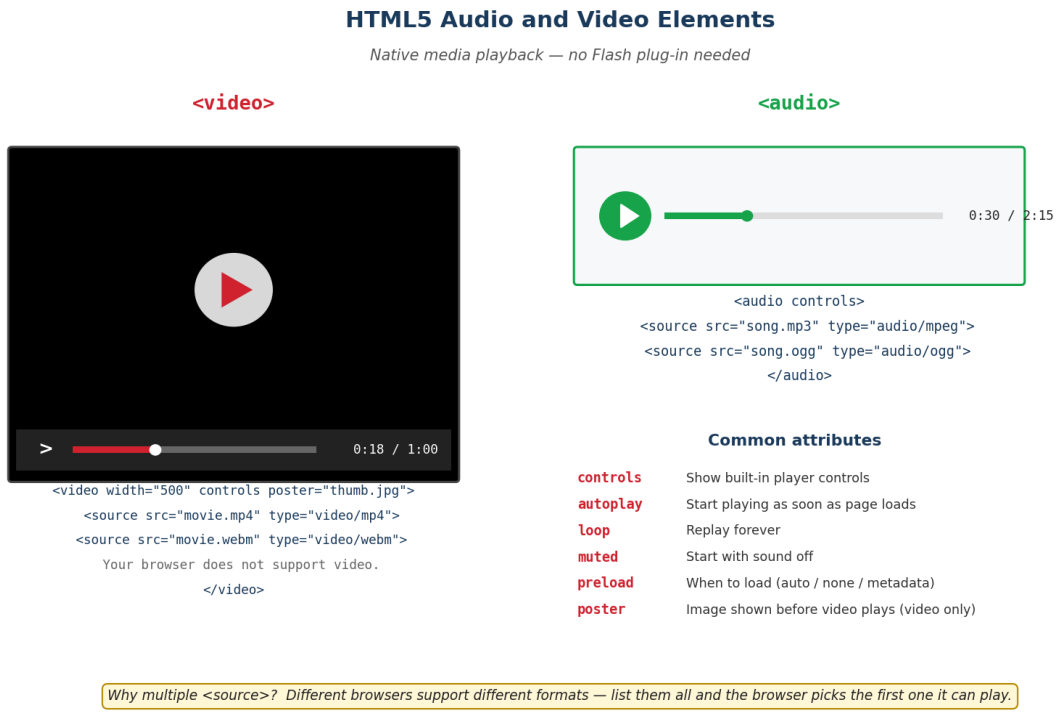


Figure 4.3 — Side-by-side anatomy of <audio> and <video> elements with common attributes.

8.2 Video Attributes

Tag / Attribute	Purpose
width / height	Size of the video player in pixels.
poster	Image to display before the video starts (e.g. a thumbnail).
controls	Show the player controls (play, pause, volume, fullscreen).
autoplay	Start playing automatically (most browsers require muted="true" too).
loop	Replay forever.
muted	Start with sound off.
playsinline	On mobile, play inside the page instead of full-screen.

8.3 Common Video Formats

Format	File Extension	Notes
MP4 (H.264)	.mp4	Most widely supported — the safe default.
WebM	.webm	Open-source, smaller files, Chrome/Firefox/Edge.
Ogg Theora	.ogv	Open-source, older format.

Why three <source> tags?
Different browsers support different formats. Listing several sources lets the browser pick the first one it can play. If none work, the fallback text inside <video>...</video> is shown.

9. Form Helper Elements

9.1 <label> — Always Use Labels

Every input should have a label so users know what to type. The label is linked to its input using the for attribute (which matches the input's id). Clicking on the label focuses the input — very useful on mobile.

Example 9.1 — Label-input pair

```
<label for="name">Full Name:</label>
<input type="text" id="name" name="fullname">
```

9.2 <fieldset> and <legend> — Grouping

Long forms become easier to read when related fields are grouped. <fieldset> draws a border around the group; <legend> is the group's title.

Example 9.2 — Grouping fields with fieldset and legend

```
<fieldset>
  <legend>Contact Information</legend>

  <label for="em">Email:</label>
  <input type="email" id="em" name="email">

  <label for="ph">Phone:</label>
  <input type="tel" id="ph" name="phone">
</fieldset>
```

9.3 <datalist> — Auto-Suggestions

<datalist> gives the user a list of suggestions while still allowing them to type a custom value. It is like a smart, free-form dropdown.

Example 9.3 — Smart text input with five city suggestions

```
<label for="city">City:</label>
<input list="cities" id="city" name="city">

<datalist id="cities">
  <option value="Jamshedpur">
  <option value="Ranchi">
  <option value="Patna">
  <option value="Kolkata">
  <option value="Delhi">
</datalist>
```

10. Worked Example — Complete Registration Form

Let's combine everything from this week into one complete BCA admission registration form with proper grouping, validation, and an embedded video for the welcome message.

Example 10.1 — Complete BCA admission form with video, validation, and grouping

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>BCA Admission Registration | RVSCET</title>
</head>
<body>

  <header>
```

```
    <h1>BCA Admission 2026 — RVSCET</h1>
    <p>Watch the welcome message before filling the form:</p>
    <video width="500" controls poster="thumb.jpg">
      <source src="welcome.mp4" type="video/mp4">
      Your browser does not support video.
    </video>
</header>

<form action="register.php" method="POST">

    <fieldset>
      <legend>Personal Details</legend>

      <label for="name">Full Name:</label>
      <input type="text" id="name" name="name"
           required minlength="3" maxlength="50"><br><br>

      <label for="dob">Date of Birth:</label>
      <input type="date" id="dob" name="dob" required><br><br>

      <label>Gender:</label>
      <input type="radio" name="gender" value="M" required> Male
      <input type="radio" name="gender" value="F"> Female
      <input type="radio" name="gender" value="O"> Other
      <br><br>
    </fieldset>

    <fieldset>
      <legend>Contact Information</legend>

      <label for="email">Email:</label>
      <input type="email" id="email" name="email" required><br><br>

      <label for="mob">Mobile:</label>
      <input type="tel" id="mob" name="mobile"
           pattern="[6-9][0-9]{9}"
           title="10 digits, starts with 6/7/8/9" required><br><br>

      <label for="addr">Address:</label><br>
      <textarea id="addr" name="address" rows="3" cols="40"
            required></textarea><br><br>
    </fieldset>

    <fieldset>
      <legend>Academic Info</legend>

      <label for="marks">12th %:</label>
      <input type="number" id="marks" name="marks"
```

```
            min="0" max="100" step="0.01" required><br><br>

        <label for="stream">Stream:</label>
        <select id="stream" name="stream" required>
          <option value="">-- Select --</option>
          <option value="science">Science</option>
          <option value="commerce">Commerce</option>
          <option value="arts">Arts</option>
        </select><br><br>

        <label>Hobbies:</label>
        <input type="checkbox" name="hobbies" value="coding"> Coding
        <input type="checkbox" name="hobbies" value="music"> Music
        <input type="checkbox" name="hobbies" value="sports"> Sports
        <br><br>

        <label for="photo">Upload Photo:</label>
        <input type="file" id="photo" name="photo"
            accept="image/*" required><br><br>
      </fieldset>

      <button type="submit">Register</button>
      <button type="reset">Clear</button>

    </form>

</body>
</html>
```

Try this in your lab

1. Save the code as register.html.
2. Open it in Chrome / Firefox / Edge.
3. Try submitting with empty fields — see how the browser blocks you.
4. Type an invalid email — see the helpful error message.
5. Try a 9-digit mobile number — the pattern attribute will reject it.
6. Inspect what happens when you click Submit. (Without a real PHP server, the browser will just go to register.php and show "file not found" — that's expected. We'll connect it to a real backend in Week 10.)

11. Summary of Week 4

- A form sends user data to a server using <form action="..." method="...">.
- GET puts data in the URL (good for searches); POST hides it in the body (good for sensitive data).
- Common controls: <input>, <select>, <textarea>, and <button>.

- HTML5 added rich input types — email, url, tel, number, range, date, time, color, file, search.
- Built-in validation uses required, pattern, min, max, minlength, maxlength — no JavaScript needed.
- Always pair every input with a <label>, and group related inputs with <fieldset>+<legend>.
- Audio is added with <audio>, video with <video> — both support the controls attribute and multiple <source> tags for cross-browser compatibility.
- Browser-side validation is for user friendliness; server-side validation is mandatory for security.

12. Practice Questions

A. Short Answer (2 marks each)

1. What is the difference between GET and POST methods? Give one example of each.

2. Differentiate between radio buttons and checkboxes with a code example.

3. Why is the name attribute required on every form input?

4. What is the purpose of the <label> tag and how is it linked to an input?

5. List any five new input types introduced in HTML5.

6. Write the HTML for a number field that accepts marks between 0 and 100, with a default of 50.

7. What does the controls attribute do in <audio> and <video>?

B. Long Answer (5–10 marks each)

1. Explain the structure of an HTML5 form with action, method, and at least four different controls. Use a diagram.

2. Describe at least eight new HTML5 input types with code examples and explain when each is useful.

3. Explain HTML5 input validation using required, pattern, min, max, minlength, and maxlength with examples.

4. Compare the <audio> and <video> elements: similarities, differences, and common attributes. Give one full example of each.

5. Why are <fieldset>, <legend>, <label>, and <datalist> useful in a form? Give an example using all four.

C. Lab / Hands-On Tasks

1. Create a contact form with name, email, mobile, subject, and message. Apply HTML5 validation so that all five fields are required and email/mobile follow correct formats.

2. Build a feedback form for the BCA Department with: name (text), department (dropdown), rating (range from 1-10), suggestions (textarea), and a Submit button.

3. Make a simple media player page with one <audio> and one <video>, both with custom poster image (for video) and three <source> formats each.

4. Design an exam application form using all the new HTML5 input types covered this week (email, tel, number, date, color, range, file). Group fields using <fieldset> with proper <legend> titles.

5. Create a course registration form that uses <datalist> to suggest courses while still allowing the user to type a new course name.

WEEK 5 — CSS3 BASICS

Course Outcome: CO2 / CO3 | Topics: CSS syntax, selectors, colors, fonts, text styling, the box model

Learning Objectives

By the end of this week, students will be able to:

- Explain what CSS is, why it is used, and how it relates to HTML.
- Add CSS to a page in three different ways — inline, internal, and external.
- Write CSS rules using element, class, and id selectors.
- Style text using color, font, size, alignment, and decoration properties.
- Style images using border, width, height, and basic effects.
- Apply the CSS box model — content, padding, border, and margin — to control spacing and layout.

1. Introduction to CSS

1.1 What is CSS?

CSS stands for Cascading Style Sheets. It is the language that tells the browser how an HTML page should look — colours, fonts, spacing, layout, and effects. HTML provides the content (words, images, links), and CSS provides the appearance (the look and feel).

Think of HTML as the bricks of a building and CSS as the paint, wallpaper, and furniture. The same HTML page can look completely different just by changing its CSS.

1.2 Why Separate HTML and CSS?

- Cleaner code — HTML stays focused on structure, CSS on style.
- Reusability — one CSS file can style hundreds of HTML pages.
- Easier maintenance — change a colour in one place, every page updates.
- Faster page loads — the browser caches CSS files.
- Better teamwork — designers work on CSS while developers write HTML.

1.3 A Quick Comparison

Aspect	HTML	CSS
Purpose	Defines the content and structure of the page.	Defines how the content looks on screen.
File type	.html	.css
Tags / Syntax	<h1>, <p>, <img>, <a>, ...	selector { property: value; }
Example	Hello	h1 { color: blue; }

1.4 CSS vs CSS3

CSS3 is the third major version of CSS — the one used by every modern browser today. It added many powerful new features: rounded corners, shadows, gradients, transitions, animations, flexbox, grid, and media queries. We will cover the basics this week and the advanced features in Weeks 6 and 7.

2. CSS Syntax

2.1 The Basic Rule

Every CSS rule has the same structure: a selector that says "which element", followed by a list of declarations inside curly braces. Each declaration is a property : value pair ending with a semicolon.

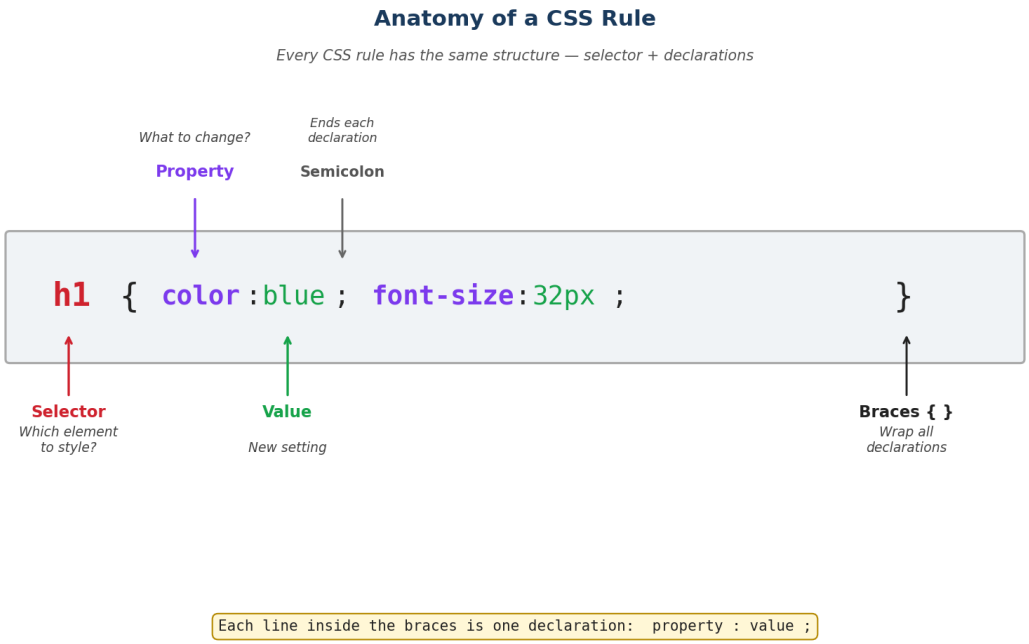


Figure 5.1 — Anatomy of a CSS rule with every part labelled.

Example 2.1 — One rule with three declarations

```
h1 {
   color: blue;
   font-size: 32px;
   text-align: center;
}
```

2.2 Comments in CSS

Comments help you remember why you wrote a rule. They are ignored by the browser.

Example 2.2 — CSS comments

```
/* This is a single-line CSS comment */

/* Comments can also span
```

```
  across multiple lines */

h1 {
   color: red;       /* main heading colour */
   font-size: 28px;  /* a bit smaller than default */
}
```

Remember

1. Selectors come first, then { } braces.
2. Every declaration ends with a semicolon ;
3. Properties and values are separated by a colon :
4. Use lowercase for property names; values like colours can be in any case.

3. Three Ways to Add CSS to a Page

There are three places where CSS can live: directly inside a tag (inline), inside the <head> section (internal), or in a separate file (external). Each has its uses, but external CSS is the standard professional choice.

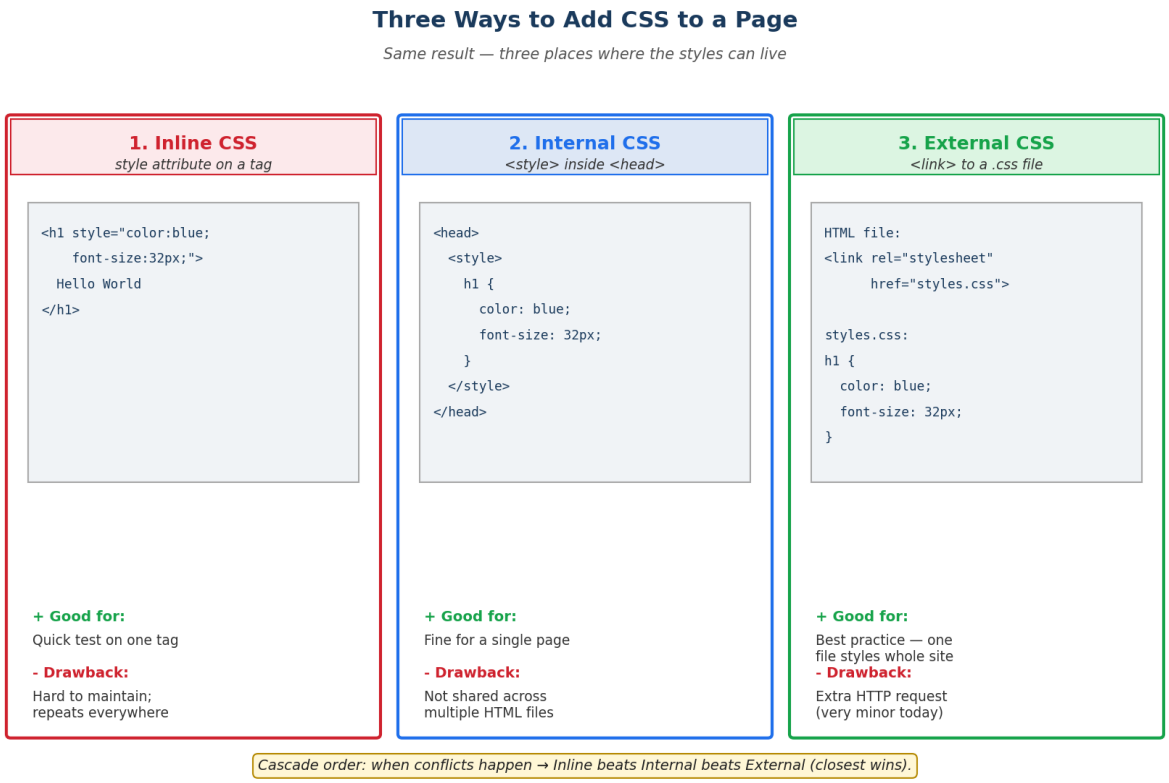


Figure 5.2 — The three ways to add CSS, side by side, with pros and cons.

3.1 Inline CSS — using the style attribute

Example 3.1 — Inline CSS applied to one tag

```
<h1 style="color: blue; font-size: 32px;">
   Welcome to RVSCET
</h1>

<p style="color: green;">This paragraph is green.</p>
```

3.2 Internal CSS — using a <style> block

Example 3.2 — Internal CSS inside the <head>

```
<!DOCTYPE html>
<html>
<head>
   <title>Internal CSS Demo</title>
   <style>
      h1 {
         color: blue;
         font-size: 32px;
      }
      p {
         color: #333;
         line-height: 1.6;
      }
   </style>
</head>
<body>
   <h1>Welcome to RVSCET</h1>
   <p>This page uses internal CSS.</p>
</body>
</html>
```

3.3 External CSS — using a separate .css file

This is the recommended approach. Create one CSS file and link it from any number of HTML pages.

Example 3.3a — The HTML file links to an external CSS

```
<!-- File 1: index.html -->
<!DOCTYPE html>
<html>
<head>
   <title>External CSS Demo</title>
   <link rel="stylesheet" href="styles.css">
</head>
<body>
   <h1>Welcome to RVSCET</h1>
   <p>Styles come from styles.css</p>
</body>
```

```
</html>
```

Example 3.3b — The matching styles.css file

```
/* File 2: styles.css */

h1 {
    color: blue;
    font-size: 32px;
    text-align: center;
}

p {
    color: #333;
    line-height: 1.6;
    margin: 10px;
}
```

3.4 Which One to Use?

Method	Best Used For	Avoid When
Inline	Quick test on one element; emails (HTML emails).	Real websites — repeats everywhere.
Internal	Single small page or one-off demo.	Multiple pages share styles.
External	Real websites, multiple pages, professional projects.	Almost never — this is the default choice.

The Cascade — what wins when rules conflict?

If two rules both target the same element, the closer one wins:

Inline (in the tag) beats Internal (in the head) beats External (in a .css file).

Within the same file, a later rule beats an earlier one with equal specificity.

The word "Cascading" in CSS comes from this top-to-bottom flow of priority.

4. CSS Selectors

4.1 What is a Selector?

A selector tells the browser which HTML elements to style. CSS gives many ways to pick elements — by tag name, by class, by id, or by relationship. We will start with the three you will use most: element, class, and id.

4.2 Element (Tag) Selector

Targets every element of a given tag name on the page.

Example 4.1 — Element selectors

```
/* Style every <h1> in the page */
h1 {
  color: navy;
}

/* Style every paragraph */
p {
  line-height: 1.5;
}

/* Style every link */
a {
  color: #1f6feb;
}
```

4.3 Class Selector

A class selector targets any element that has a matching class attribute. Many elements can share the same class. In CSS, a class is written with a leading dot (.).

Example 4.2 — A class targets multiple elements

```
<!-- HTML -->
<p class="highlight">Important notice</p>
<p>Normal paragraph</p>
<h2 class="highlight">Important heading</h2>

/* CSS */
.highlight {
  background-color: yellow;
  font-weight: bold;
}
```

4.4 ID Selector

An id selector targets the one element with a matching id attribute. An id must be unique on a page. In CSS, an id is written with a leading hash (#).

Example 4.3 — An id targets one specific element

```
<!-- HTML -->
<header id="main-header">
  <h1>RVSCET</h1>
</header>

/* CSS */
#main-header {
  background-color: navy;
```

```
    color: white;
    padding: 20px;
}
```

4.5 Class vs. ID — Quick Comparison

Aspect	Class (.)	ID (#)
Reusable?	Yes — many elements can share a class.	No — each id is unique on the page.
CSS prefix	. (dot)	# (hash)
HTML attribute	class="name"	id="name"
Best used for	Reusable styles (button, card, alert).	One unique element (header, sidebar).

4.6 Grouping Selectors

If several selectors share the same styles, write them as a comma-separated list.

Example 4.4 — Grouping multiple selectors

```
/* Three selectors share the same colour */
h1, h2, h3 {
    color: navy;
    font-family: 'Segoe UI', sans-serif;
}
```

4.7 Universal Selector

The asterisk * targets every element on the page. It is most often used to reset the default browser margins.

Example 4.5 — The universal selector resetting defaults

```
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
```

5. Colors in CSS

5.1 Five Ways to Specify a Colour

Example 5.1 — Six ways to write the same kind of blue

```
h1 { color: red; }                    /* 1. Named colour      */
```

```
h2 { color: #1F6FEB; }                  /* 2. Hex code (6 digits) */
h3 { color: #FFF; }                     /* 3. Short hex (3 digits) */
h4 { color: rgb(31, 111, 235); }        /* 4. RGB                  */
h5 { color: rgba(31, 111, 235, 0.5); }  /* 5. RGBA (with alpha)  */
h6 { color: hsl(213, 84%, 52%); }       /* 6. HSL                  */
```

Format	What it Means
Named	About 140 named colours: red, green, blue, navy, tomato, gold, lavender, ...
#RRGGBB	Hexadecimal — two digits for Red, Green, Blue (00 to FF).
#RGB	Short hex — each digit is doubled. #F00 means #FF0000.
rgb(r,g,b)	Red, green, blue values from 0 to 255.
rgba(r,g,b,a)	Same as rgb plus alpha (0 = transparent, 1 = opaque).
hsl(h,s,l)	Hue (0–360°), saturation, lightness — easier for designers.

5.2 Foreground and Background

Example 5.2 — Setting background and foreground colours

```
body {
  background-color: #f4f6f8;   /* page background */
  color: #222;                 /* default text colour */
}

.alert {
  background-color: #fff7d6;
  color: #7a5500;
  padding: 12px;
}
```

> ***Output in browser:***

(The page has a light grey background.)

- A yellow alert box with dark text.

5.3 A Mini Colour Palette for the BCA Site

Use	Colour	Code
Primary brand	Navy	#1A3A5C
Accent	Bright blue	#1F6FEB
Success / OK	Green	#16A34A

Use	Colour	Code
Warning	Amber	#F59E0B
Error	Red	#DC2626
Page background	Off-white	#F4F6F8

6. Fonts and Text

6.1 font-family

The font-family property sets the typeface. List several fonts as a fallback chain — the browser uses the first one available on the user's system.

Example 6.1 — Three font stacks for body, code, and a fancy heading

```
body {
  font-family: 'Segoe UI', Arial, Helvetica, sans-serif;
}

code {
  font-family: 'Consolas', 'Courier New', monospace;
}

h1.fancy {
  font-family: 'Georgia', 'Times New Roman', serif;
}
```

6.2 Font Categories

Family	Look	Common Examples
serif	Has small "feet" on letters.	Times New Roman, Georgia, Cambria
sans-serif	Clean, no feet — modern look.	Arial, Helvetica, Segoe UI, Roboto
monospace	Every letter is the same width.	Consolas, Courier New, Fira Code
cursive	Looks like handwriting.	Brush Script, Comic Sans
fantasy	Decorative and unusual.	Impact, Papyrus

6.3 Font Size, Weight, and Style

Example 6.2 — Font size, weight, and style

```
h1 {
```

```
    font-size: 32px;
    font-weight: bold;      /* normal, bold, 100..900 */
    font-style: normal;     /* normal, italic, oblique */
}

p {
    font-size: 1rem;        /* 1rem = root font size, usually 16px */
    font-weight: 400;
}

.note {
    font-size: 0.9em;       /* relative to parent */
    font-style: italic;
}
```

Unit	Meaning
px	Pixels — fixed size. Predictable but does not scale with user settings.
em	Relative to the font-size of the parent element. 2em = double the parent.
rem	Relative to the root font-size (the <html> element). Easier to manage.
%	Relative to the parent element.
pt	Points — used in print, rarely in screens (1pt is about 1.33px).

6.4 Text Properties

Example 6.3 — Common text-related properties

```
p {
    color: #222;
    text-align: justify;         /* left, right, center, justify */
    text-decoration: none;       /* underline, overline, line-through */
    text-transform: capitalize;  /* uppercase, lowercase, capitalize */
    line-height: 1.6;            /* spacing between lines */
    letter-spacing: 0.5px;       /* space between letters */
    word-spacing: 2px;           /* space between words */
    text-indent: 30px;           /* first-line indent */
}
```

Property	What it does
text-align	Horizontal alignment: left, right, center, justify.
text-decoration	Underline / overline / line-through / none.
text-transform	uppercase / lowercase / capitalize.

line-height	Vertical spacing between lines (try 1.4 to 1.8 for body text).
letter-spacing	Extra space between letters.
word-spacing	Extra space between words.
text-indent	Indents the first line of a paragraph.
text-shadow	Adds a shadow behind the text.

6.5 A Worked Text-Styling Example

Example 6.4 — A complete text-styling example

```
<style>
  h1 {
    font-family: 'Segoe UI', sans-serif;
    color: #1a3a5c;
    text-align: center;
    text-transform: uppercase;
    letter-spacing: 2px;
  }
  p {
    font-family: Georgia, serif;
    font-size: 18px;
    line-height: 1.7;
    color: #333;
    text-align: justify;
    text-indent: 30px;
  }
</style>

<h1>Welcome to BCA</h1>
<p>The Bachelor of Computer Applications programme at RVSCET
  prepares students for the IT industry with hands-on training
  and modern curriculum.</p>
```

7. Styling Images

7.1 Width and Height

Example 7.1 — Sizing images

```
img {
  width: 300px;      /* fixed width */
  height: auto;      /* keep aspect ratio */
}

.thumb {
```

```
    width: 150px;
    height: 150px;
    object-fit: cover;  /* crop to fill the box */
}

.responsive {
    max-width: 100%;    /* never wider than its container */
    height: auto;
}
```

7.2 Borders and Rounded Corners

Example 7.2 — Borders and rounded corners on images

```
.profile {
    width: 150px;
    height: 150px;
    border: 4px solid #1f6feb;
    border-radius: 50%;     /* circular image */
}

.card-image {
    border: 1px solid #ccc;
    border-radius: 8px;     /* slightly rounded corners */
}
```

7.3 Other Useful Image Properties

Property	Effect
opacity	Transparency from 0 (invisible) to 1 (fully visible).
box-shadow	Drops a shadow around the image.
filter	Photo effects: blur(), grayscale(), brightness(), contrast().
object-fit	How an image fills its box: cover, contain, fill, none.

Example 7.3 — Image effects with one CSS property each

```
.faded     { opacity: 0.6; }
.shadowed  { box-shadow: 4px 4px 12px rgba(0, 0, 0, 0.3); }
.grey      { filter: grayscale(100%); }
.bright    { filter: brightness(1.2); }
.blurred   { filter: blur(3px); }
```

8. The CSS Box Model

8.1 Every Element is a Box

In CSS, every HTML element on the page is treated as a rectangular box. Each box has four layers, working from the inside out: content, padding, border, and margin. Understanding the box model is the key to laying out a page correctly.

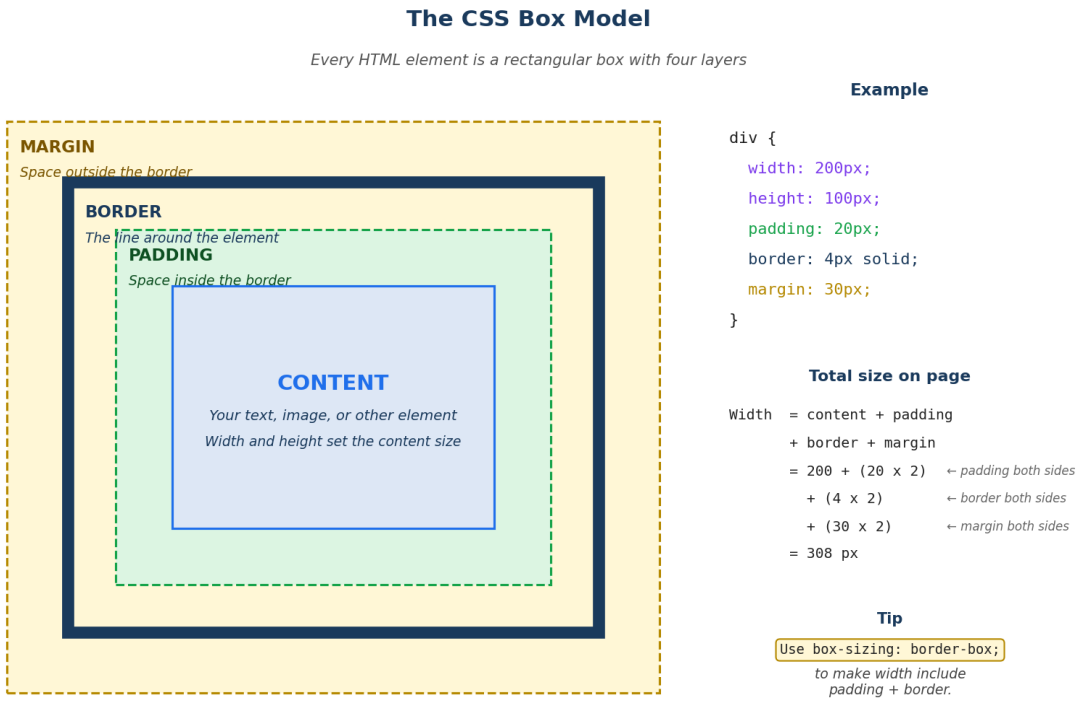


Figure 5.3 — The four layers of the CSS box model with an example calculation.

8.2 The Four Layers

Layer	What it represents
content	The actual text, image, or element. Its size is set by width and height.
padding	Space between the content and the border. Padding takes the element's background colour.
border	A line drawn around the padding. Has thickness, style, and colour.
margin	Empty space outside the border, separating this element from its neighbours.

8.3 Box Model Properties

Example 8.1 — All box-model properties applied to one element

```
.box {
  /* Content size */
  width: 300px;
```

```
    height: 150px;

    /* Padding (inside the border) */
    padding: 20px;                  /* same on all four sides */
    /* OR set each side separately */
    /* padding-top: 10px;                                  */
    /* padding-right: 20px;                                */
    /* padding-bottom: 10px;                                  */
    /* padding-left: 20px;                                */
    /* OR shorthand: top right bottom left */
    /* padding: 10px 20px 10px 20px;                          */

    /* Border */
    border: 2px solid #1f6feb;          /* width style colour */
    border-radius: 8px;                 /* rounded corners */

    /* Margin (outside the border) */
    margin: 30px;
}
```

8.4 Border Styles

Example 8.2 — Different border styles

```
.b1 { border: 2px solid  red; }
.b2 { border: 2px dashed blue; }
.b3 { border: 2px dotted green; }
.b4 { border: 4px double navy; }
.b5 { border: 2px groove orange; }
.b6 { border: 0; }                  /* no border */
```

8.5 Calculating the Total Size

By default, the width and height properties only set the size of the content area. The padding, border, and margin add to that. Look at this example:

Example 8.3 — A simple card

```
.card {
    width: 200px;
    height: 100px;
    padding: 20px;
    border: 4px solid;
    margin: 30px;
}
```

The total horizontal space taken by this card on the page is:

Calculation

```
Total width = content + padding + border + margin
            = 200 + (20 x 2) + (4 x 2) + (30 x 2)
            = 200 + 40 + 8 + 60
            = 308 px
```

Tip — box-sizing: border-box

If you want width and height to include padding and border (so a 200px card stays exactly 200px wide), add this rule at the top of your CSS:

* { box-sizing: border-box; }

Most modern websites use this. It makes layout much more predictable.

9. Worked Example — Styled BCA Department Page

Let's apply everything from this week to the BCA Department home page from Week 3. We'll keep the HTML simple and add an external CSS file.

Example 9.1a — bca.html

```
<!-- File: bca.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>BCA Department | RVSCET</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

  <header id="top">
    <h1>RVSCET — BCA Department</h1>
    <p class="tagline">Empowering tomorrow's developers, today.</p>
  </header>

  <main>
    <section class="card">
      <h2>Welcome to BCA</h2>
      <p>The Bachelor of Computer Applications programme at
        RVSCET, Jamshedpur is a three-year degree designed to
        prepare students for the IT industry.</p>
    </section>

    <section class="card highlight">
      <h2>Apply Now</h2>
      <p>Admissions for 2026 are open. Limited seats available.</p>
    </section>
  </main>
```

```
    <footer>
        <p>&copy; 2026 RVSCET, Jamshedpur</p>
    </footer>

</body>
</html>
```

Example 9.1b — styles.css

```
/* File: styles.css */

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Arial, sans-serif;
    background-color: #f4f6f8;
    color: #222;
    line-height: 1.6;
}

#top {
    background-color: #1a3a5c;
    color: white;
    padding: 30px;
    text-align: center;
}

#top h1 {
    font-size: 32px;
    text-transform: uppercase;
    letter-spacing: 2px;
}

.tagline {
    font-style: italic;
    color: #cfd8e3;
    margin-top: 8px;
}

main {
    max-width: 800px;
    margin: 30px auto;        /* centred horizontally */
    padding: 20px;
}
```

```
.card {
    background-color: white;
    border: 1px solid #ddd;
    border-radius: 8px;
    padding: 20px;
    margin-bottom: 20px;
    box-shadow: 0 2px 6px rgba(0, 0, 0, 0.05);
}

.card h2 {
    color: #1f6feb;
    margin-bottom: 10px;
}

.highlight {
    background-color: #fff7d6;
    border-color: #f59e0b;
}

footer {
    text-align: center;
    padding: 20px;
    background-color: #1a3a5c;
    color: white;
    margin-top: 40px;
}
```

Try this on your computer

1. Save bca.html and styles.css in the same folder.
2. Double-click bca.html — it will open in your browser.
3. Try changing the colours in styles.css and refresh the page.
4. Notice how the same HTML looks completely different just by editing the CSS file.

10. Summary of Week 5

- CSS controls the look of HTML pages — colours, fonts, spacing, and layout.
- A CSS rule has the form: selector { property: value; ... }.
- CSS can be added inline, internally (in <style>), or externally (in a .css file). External is the professional choice.
- Selectors target elements: tag selector (h1), class selector (.box), id selector (#main).
- Colours are written as named (red), hex (#1F6FEB), rgb(), rgba(), or hsl() values.
- Fonts are set with font-family — list several as fallback. Use font-size, font-weight, and font-style to control appearance.

- Text properties include text-align, text-decoration, text-transform, line-height, letter-spacing, and text-shadow.
- Images can be sized, bordered, rounded (border-radius), shadowed, and filtered.
- The box model wraps every element in four layers: content + padding + border + margin.
- Use box-sizing: border-box for predictable widths.

11. Practice Questions

A. Short Answer (2 marks each)

1. What does CSS stand for and why is it used?

2. Write the syntax of a CSS rule and label each part.

3. Differentiate between inline, internal, and external CSS with one example each.

4. What is the difference between a class selector and an id selector?

5. List five different ways to write the colour blue in CSS.

6. Explain the four layers of the CSS box model with a labelled diagram.

7. Write a CSS rule that makes all <h1> elements navy, centered, and uppercase.

B. Long Answer (5–10 marks each)

1. Explain the three ways to add CSS to an HTML page with code examples and discuss when each method is appropriate.

2. Describe the most common CSS selectors (element, class, id, group, universal) with a code example for each.

3. Explain font-family, font-size, font-weight, and at least four text properties with examples.

4. Describe the CSS box model and calculate the total horizontal size of an element with width 250px, padding 15px, border 3px, and margin 20px.

5. Discuss the cascade rule in CSS. What happens when an inline style, an internal style, and an external style all target the same element?

C. Lab / Hands-On Tasks

1. Create an HTML page about your favourite movie. Use external CSS to style the heading (centered, uppercase, navy colour), the paragraphs (justified, 1.6 line-height), and add a coloured background to the body.

2. Build a profile card for yourself — name, photo (rounded), role, and a short bio — using a class .card with padding, border, and rounded corners. Add a box-shadow effect.

3. Create a styled timetable using HTML tables and CSS. Apply alternating row colours, a coloured header row, and padding to all cells.

4. Create a webpage with three different paragraphs, each styled with a different class (.normal, .warning, .success) using contrasting background and text colours.

5. Take the Week 3 BCA Department page and rewrite all its appearance using a single external styles.css file. Use at least one id selector, three class selectors, and demonstrate the box model.

WEEK 6 — CSS3 LAYOUT & RESPONSIVENESS

Course Outcome: CO3 | Topics: display, position, float, Flexbox, CSS Grid, media queries

Learning Objectives

By the end of this week, students will be able to:

- Explain how CSS controls the layout of a web page.
- Use the display property — block, inline, inline-block, and none.
- Use the position property — static, relative, absolute, fixed, and sticky.
- Build flexible one-dimensional layouts using Flexbox.
- Build two-dimensional layouts using CSS Grid.
- Make a page responsive using media queries and the meta viewport tag.
- Combine all of the above to build a multi-section, mobile-friendly web page.

1. Introduction to Layout in CSS

1.1 What is Layout?

Layout means deciding where each element appears on the page — whether headings sit on top, the menu is on the left, the content is in the middle, and the footer is at the bottom. CSS gives us several tools to control layout — display, position, float, Flexbox, and Grid. Each tool is best for a different situation.

1.2 How CSS Decides Where Things Go

By default, the browser places HTML elements in normal flow — top to bottom, one after another. CSS lets us break out of this flow when needed: stack things in rows, place items in a grid, pin a header to the top of the screen, or float an image to the side of a paragraph.

> **From Week 5 — quick recap**
> Every element is a rectangular box with content + padding + border + margin (the box model).
> Selectors (#id, .class, tag) decide which boxes get which styles.
> This week we focus on positioning those boxes — placing them, sizing them, and arranging them on screen.

2. The display Property

2.1 Block, Inline, and Inline-Block

Every HTML element has a default display behaviour. Block elements take a full line; inline elements only take as much space as their content. The display property lets us override this default.

Value	Behaviour	Examples
block	Full line; new line before and after; width and height work.	div, p, h1–h6, section

Value	Behaviour	Examples
inline	Flows in line with text; width and height ignored.	span, a, strong, em
inline-block	Flows like inline but width and height work.	Custom buttons, badges
none	Hidden — element disappears from the page completely.	Hidden menus, modals

Example 2.1 — Using the display property

```
/* Make every <li> sit on the same line */
nav li {
  display: inline-block;
  padding: 8px 16px;
}

/* Hide an alert box completely */
.alert.dismissed {
  display: none;
}

/* Turn a span into a block-level element */
.badge {
  display: inline-block;
  width: 80px;
  background-color: #1f6feb;
  color: white;
  text-align: center;
  border-radius: 12px;
}
```

Tip — visibility: hidden vs display: none

display: none — element is removed; the space it took is gone.

visibility: hidden — element is invisible but still occupies its space (a blank gap remains).

3. The position Property

3.1 Five Position Values

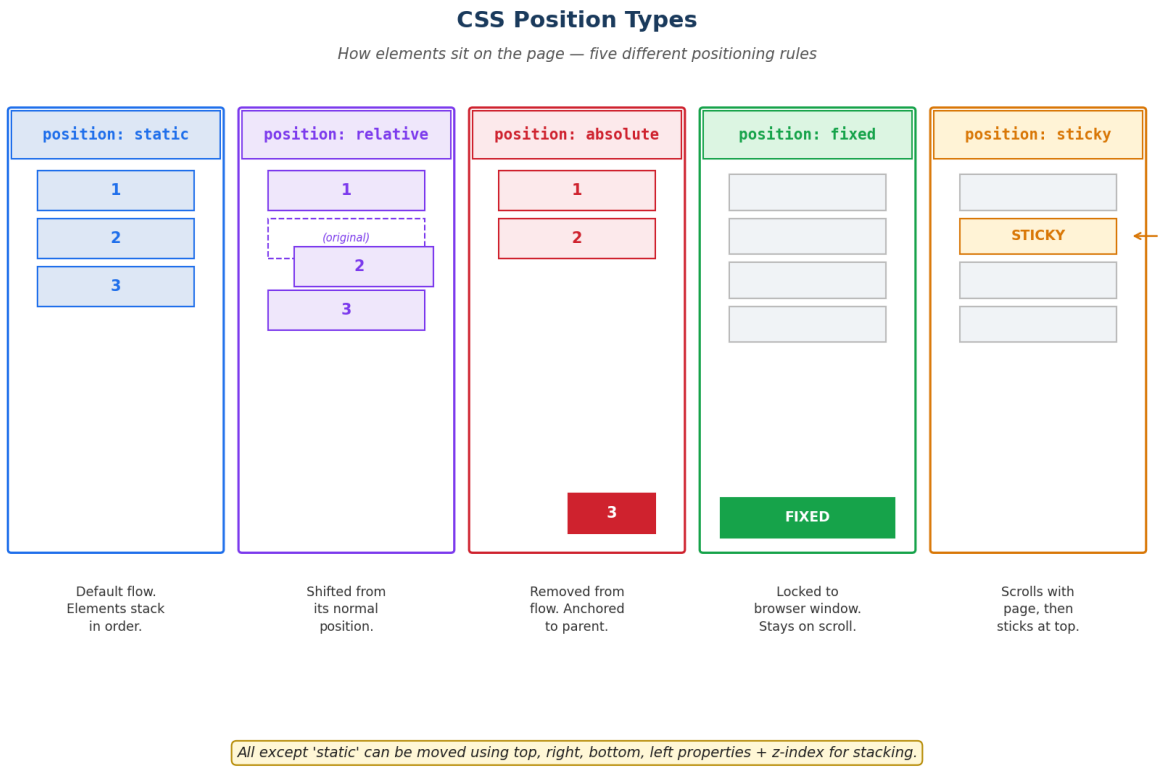


Figure 6.1 — The five CSS position values, side by side, with how each behaves on the page.

Value	What it does
static	Default — element sits in normal document flow. top, right, bottom, left have NO effect.
relative	Element sits in normal flow, then is shifted by top/right/bottom/left from its original spot.
absolute	Removed from flow. Pinned to the nearest positioned ancestor (or the page if none).
fixed	Removed from flow. Pinned to the browser window — stays put when the page scrolls.
sticky	Behaves like relative, but sticks to a position once you scroll past it.

3.2 Examples

Example 3.1 — All five position values in action

```
/* relative: shift a heading 20px to the right */
.tag {
   position: relative;
   left: 20px;
}
```

```
/* absolute: pin a small badge to the top-right of its parent card */
.card {
   position: relative;       /* parent must NOT be static */
}
.card .ribbon {
   position: absolute;
   top: 10px;
   right: 10px;
   background: red;
   color: white;
   padding: 4px 8px;
}

/* fixed: keep a navigation bar at the top of the screen */
header {
   position: fixed;
   top: 0;
   left: 0;
   width: 100%;
   background: #1a3a5c;
   color: white;
}

/* sticky: a sub-heading that sticks while you scroll its section */
h2.section-title {
   position: sticky;
   top: 0;
   background: white;
}
```

3.3 z-index — Stacking Order

When two positioned elements overlap, the z-index property decides which one is on top. Higher z-index wins. Only positioned elements (anything other than static) can use z-index.

Example 3.2 — z-index controls who appears in front

```
.modal    { position: fixed; z-index: 1000; }   /* on top of everything */
.overlay  { position: fixed; z-index: 999; }    /* dim background */
.dropdown  { position: absolute; z-index: 50; }
.tooltip   { position: absolute; z-index: 100; }
```

4. The float Property

4.1 What float Does

Originally designed to wrap text around an image (like newspapers do), the float property lets an element shift to the left or right while text and inline content flow around it. Modern layouts use Flexbox or Grid for general arrangement, but float is still useful for text-image wrapping.

Example 4.1 — Floating an image with text wrap

```
/* Float a profile image to the left */
img.profile {
  float: left;
  margin: 0 15px 10px 0;
  width: 150px;
  border-radius: 8px;
}

/* Clear the float so the next section starts below */
footer {
  clear: both;
}
```

Modern advice

For layout (sidebars, columns, page structure) — use Flexbox or Grid (covered next).

Use float only for its original purpose: wrapping text around an image inside a paragraph.

5. Flexbox — One-Dimensional Layout

5.1 What is Flexbox?

Flexbox (Flexible Box Layout) makes it easy to arrange items in one dimension — either a row or a column. It is the perfect tool for navigation menus, button groups, card rows, and centring content. Just turn a container into a flex container and its children become flex items.

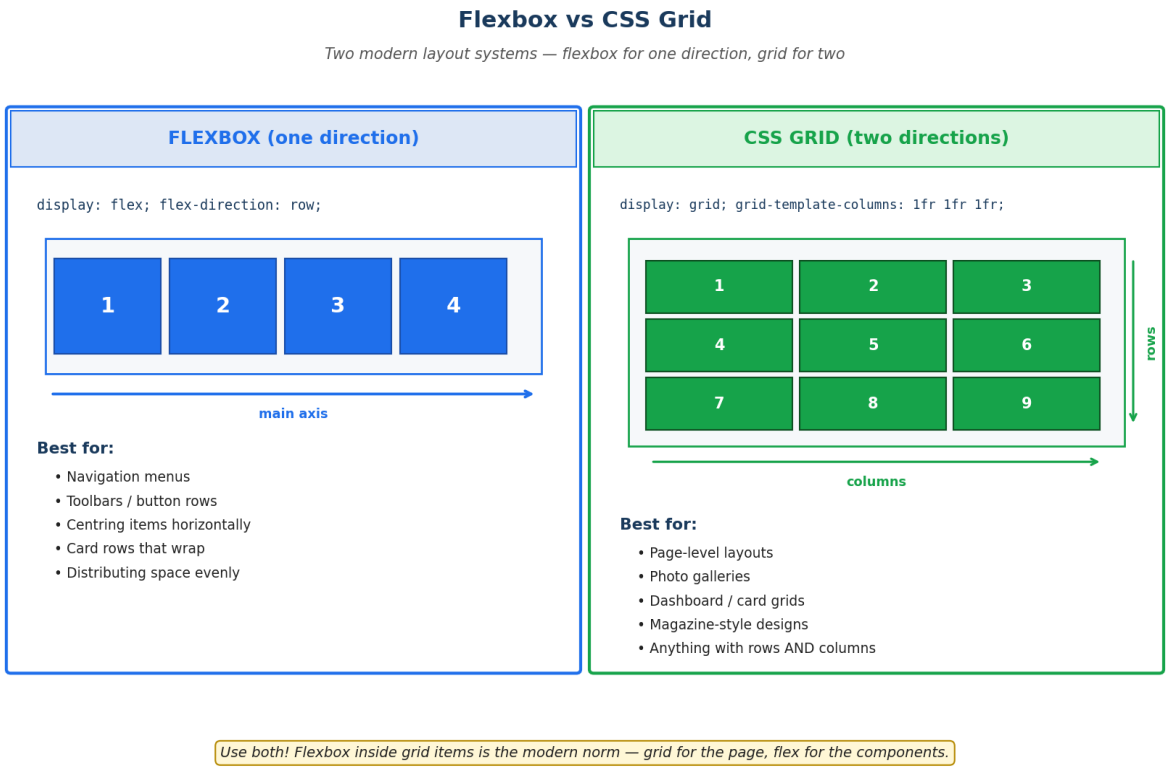


Figure 6.2 — Flexbox vs CSS Grid: when to use each system.

5.2 Turning On Flexbox

Example 5.1 — One line of CSS turns 4 links into a flex row

```
/* The parent (container) becomes a flex container */
.nav {
   display: flex;
}

<!-- HTML -->
<nav class="nav">
   <a href="#">Home</a>
   <a href="#">About</a>
   <a href="#">Courses</a>
   <a href="#">Contact</a>
</nav>
```

> ***Output in browser:***
>
> Home About Courses Contact
> (All four links sit on the same line.)

5.3 Container Properties

Property	What it does
display: flex	Turns the container into a flex container.
flex-direction	row (default), row-reverse, column, column-reverse.
justify-content	Aligns items along the main axis: flex-start, flex-end, center, space-between, space-around, space-evenly.
align-items	Aligns items along the cross axis: stretch (default), flex-start, flex-end, center, baseline.
flex-wrap	nowrap (default), wrap, wrap-reverse — should items wrap to a new line?
gap	Space between items (replaces the need for margins between flex items).

5.4 Item Properties

Property	What it does
flex-grow	How much extra space the item should take, relative to siblings (default 0).
flex-shrink	How much the item shrinks when there's not enough space (default 1).
flex-basis	The starting size of the item before grow/shrink kick in.
flex	Shorthand for grow + shrink + basis. flex: 1 is the most common — equal share.
align-self	Override the container's align-items for this single item.
order	Visual order of the item (lower = earlier). Default 0.

5.5 A Common Flexbox Pattern — Centring

Example 5.2 — The classic "centre a box on screen" trick

```
/* Centre any single item perfectly inside its container */
.centre-everything {
    display: flex;
    justify-content: center;   /* horizontal centre */
    align-items: center;       /* vertical centre */
    height: 100vh;             /* full viewport height */
}
```

5.6 A Three-Card Row

Example 5.3 — Three equal cards that wrap on small screens

```
<style>
  .cards {
    display: flex;
    gap: 20px;
    flex-wrap: wrap;
  }
  .card {
    flex: 1;              /* equal share */
    min-width: 200px;     /* don't shrink below this */
    background: white;
    border: 1px solid #ddd;
    border-radius: 8px;
    padding: 20px;
  }
</style>

<div class="cards">
  <div class="card">BCA Programme</div>
  <div class="card">MCA Programme</div>
  <div class="card">B.Tech Programme</div>
</div>
```

6. CSS Grid — Two-Dimensional Layout

6.1 What is CSS Grid?

CSS Grid lets you place elements in rows AND columns at the same time. While Flexbox handles a single line of items, Grid is designed for full page layouts — something with a header on top, a sidebar on the left, main content in the middle, and a footer at the bottom.

6.2 Defining a Grid

Example 6.1 — A 3-column photo gallery

```
.gallery {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;   /* three equal columns */
  gap: 20px;
}

<!-- HTML -->
<div class="gallery">
  <img src="1.jpg" alt="Photo 1">
  <img src="2.jpg" alt="Photo 2">
  <img src="3.jpg" alt="Photo 3">
  <img src="4.jpg" alt="Photo 4">
  <img src="5.jpg" alt="Photo 5">
```

```
    <img src="6.jpg" alt="Photo 6">
</div>
```

6.3 The fr Unit

fr means fraction. It tells the browser: "divide the available space into these proportions". 1fr 1fr 1fr means three equal columns. 2fr 1fr means the first column is twice as wide as the second.

Example 6.2 — Different ways to define grid columns

```
/* Three equal columns */
.grid1 { grid-template-columns: 1fr 1fr 1fr; }

/* Sidebar 1/3 + main content 2/3 */
.grid2 { grid-template-columns: 1fr 2fr; }

/* Fixed sidebar + flexible content + fixed sidebar */
.grid3 { grid-template-columns: 200px 1fr 200px; }

/* As many 250px columns as fit, fill remaining space equally */
.grid4 { grid-template-columns: repeat(auto-fit, minmax(250px, 1fr)); }
```

6.4 Container Properties

Property	What it does
display: grid	Turns the container into a grid container.
grid-template-columns	How many columns and how wide each one is.
grid-template-rows	Same, but for rows.
gap	Space between rows and columns.
grid-template-areas	Named areas you can place items into (advanced layout).
justify-items / align-items	Aligns each item inside its cell.

6.5 Item Properties

Example 6.3 — Making one item span multiple cells

```
.main {
    grid-column: 2 / 4;     /* span columns 2 and 3 */
    grid-row: 1 / 3;        /* span rows 1 and 2 */
}
```

6.6 Page Layout with grid-template-areas

Example 6.4 — A complete page layout in 8 lines

```
.layout {
   display: grid;
   grid-template-columns: 200px 1fr 200px;
   grid-template-rows: auto 1fr auto;
   grid-template-areas:
      "header  header  header"
      "side    main    aside"
      "footer  footer  footer";
   min-height: 100vh;
   gap: 10px;
}

header { grid-area: header; }
.side  { grid-area: side; }
main   { grid-area: main; }
aside  { grid-area: aside; }
footer { grid-area: footer; }
```

Quick rule

Use Flexbox when items go in ONE direction (a row OR a column).

Use Grid when items go in TWO directions (rows AND columns at once).

Modern sites usually use Grid for the page layout and Flexbox inside each grid section.

7. Responsive Design and Media Queries

7.1 What is Responsive Design?

Responsive design means the same web page automatically adjusts itself to look good on any screen — phone, tablet, laptop, or large monitor. Instead of building separate sites for each device, we write CSS that responds to the screen size.

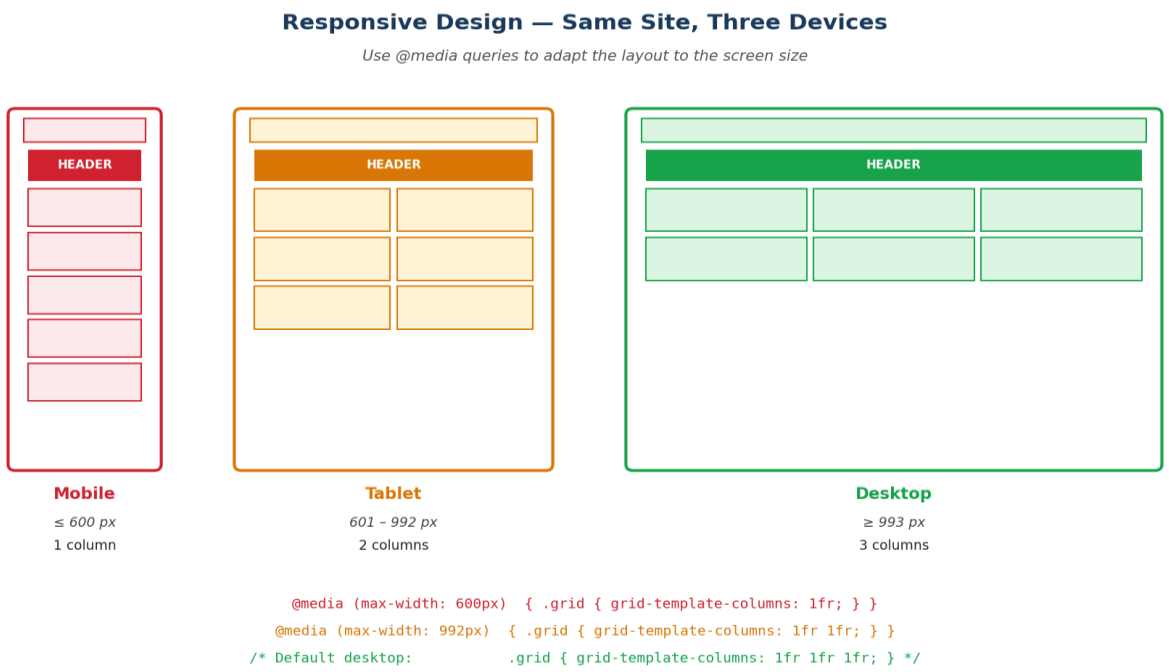


Figure 6.3 — The same site adapts from one column on mobile to three columns on desktop using media queries.

7.2 The Viewport Meta Tag

Always include this tag in the <head> of every HTML page. Without it, mobile browsers will zoom the page out to fit a desktop-sized viewport — making everything tiny and unusable.

Example 7.1 — The viewport meta tag (already in our HTML5 template)

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

7.3 Media Queries

A media query is a CSS rule wrapped in @media (...) { ... }. The styles inside only apply when the condition matches — usually a screen width range. The most common conditions are min-width and max-width.

Example 7.2 — A responsive grid using two media queries

```
/* Default styles — written for desktop first */
.grid {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;  /* 3 columns */
  gap: 20px;
}

/* When screen is 992px or narrower → switch to 2 columns */
@media (max-width: 992px) {
  .grid {
    grid-template-columns: 1fr 1fr;
  }
```

```
}

/* When screen is 600px or narrower → switch to 1 column */
@media (max-width: 600px) {
  .grid {
    grid-template-columns: 1fr;
  }
  h1 {
    font-size: 22px;        /* smaller heading on phones */
  }
}
```

7.4 Common Breakpoints

Device	Typical Width	Usual Layout
Mobile phone	≤ 600 px	1 column. Stacked vertically.
Tablet	601 – 992 px	2 columns. Sidebar collapses below.
Small desktop	993 – 1200 px	Full layout. 3 columns where used.
Large desktop	≥ 1201 px	Comfortable spacing. Capped width on content.

7.5 Mobile-First vs Desktop-First

Approach	Default styles target...	Media queries use...
Desktop-first	Big screens.	max-width — when smaller, change.
Mobile-first	Phones.	min-width — when bigger, enhance.

Mobile-first is the modern recommendation: write the simple, single-column styles first, then add media queries to add complexity for larger screens. This guarantees the site works on the smallest devices.

Example 7.3 — The mobile-first approach

```
/* Mobile-first base styles (all phones) */
.cards {
  display: grid;
  grid-template-columns: 1fr;    /* 1 column by default */
  gap: 16px;
}

/* Tablet and bigger */
```

```
@media (min-width: 601px) {
  .cards { grid-template-columns: 1fr 1fr; }
}

/* Desktop and bigger */
@media (min-width: 993px) {
  .cards { grid-template-columns: 1fr 1fr 1fr; }
}
```

8. Worked Example — A Responsive BCA Landing Page

Let's combine everything from this week — a sticky header, a Grid-based main layout, Flexbox for the navigation, and media queries for responsiveness.

Example 8.1a — index.html

```
<!-- File: index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>BCA at RVSCET</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>

  <header class="site-header">
    <h1 class="logo">RVSCET — BCA</h1>
    <nav class="main-nav">
      <a href="#home">Home</a>
      <a href="#about">About</a>
      <a href="#courses">Courses</a>
      <a href="#contact">Contact</a>
    </nav>
  </header>

  <main class="page">
    <aside class="sidebar">
      <h3>Quick Links</h3>
      <ul>
        <li><a href="#">Apply Now</a></li>
        <li><a href="#">Brochure</a></li>
        <li><a href="#">Faculty</a></li>
      </ul>
    </aside>

    <section class="content">
```

```
        <h2>Welcome to BCA at RVSCET</h2>
        <p>The Bachelor of Computer Applications programme...</p>

        <div class="cards">
          <div class="card">Web Technology</div>
          <div class="card">Data Structures</div>
          <div class="card">Mathematics</div>
          <div class="card">English</div>
          <div class="card">Programming in C</div>
          <div class="card">Database Systems</div>
        </div>
      </section>
    </main>

    <footer class="site-footer">
      <p>&copy; 2026 RVSCET, Jamshedpur</p>
    </footer>

</body>
</html>
```

Example 8.1b — styles.css

```
/* File: styles.css */

* { margin: 0; padding: 0; box-sizing: border-box; }

body {
    font-family: 'Segoe UI', Arial, sans-serif;
    color: #222;
    background: #f4f6f8;
    line-height: 1.6;
}

/* ----- Sticky header with Flexbox nav ----- */
.site-header {
    position: sticky;
    top: 0;
    z-index: 100;
    display: flex;
    justify-content: space-between;
    align-items: center;
    padding: 12px 24px;
    background: #1a3a5c;
    color: white;
    box-shadow: 0 2px 8px rgba(0,0,0,0.15);
}

.main-nav {
```

```
    display: flex;
    gap: 20px;
}

.main-nav a {
    color: white;
    text-decoration: none;
    font-weight: 500;
}

.main-nav a:hover { color: #ffd166; }

/* ----- Main page: Grid with sidebar + content ----- */
.page {
    display: grid;
    grid-template-columns: 220px 1fr;   /* sidebar + content */
    gap: 20px;
    max-width: 1200px;
    margin: 30px auto;
    padding: 0 20px;
}

.sidebar {
    background: white;
    border-radius: 8px;
    padding: 16px;
    height: fit-content;
}

.content {
    background: white;
    border-radius: 8px;
    padding: 24px;
}

/* ----- Cards: responsive auto-fit grid ----- */
.cards {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
    gap: 16px;
    margin-top: 20px;
}

.card {
    background: #1f6feb;
    color: white;
    padding: 24px;
    border-radius: 8px;
```

```
    text-align: center;
    font-weight: 600;
}

/* ----- Footer ----- */
.site-footer {
    text-align: center;
    padding: 20px;
    background: #1a3a5c;
    color: white;
    margin-top: 40px;
}

/* ----- Responsive: stack sidebar below on mobile ----- */
@media (max-width: 768px) {
    .page {
        grid-template-columns: 1fr;
    }
    .main-nav {
        gap: 12px;
        font-size: 14px;
    }
    .site-header {
        padding: 10px 16px;
        flex-direction: column;
        align-items: flex-start;
    }
}
```

Try this in your lab

1. Save both files in the same folder.
2. Open index.html in Chrome, Firefox, or Edge.
3. Resize the browser window from wide to narrow — watch the cards re-arrange and the sidebar drop below the content.
4. Open Chrome DevTools (F12) and try the device toolbar (Ctrl+Shift+M) to preview iPhone, iPad, and desktop sizes.

9. Summary of Week 6

- display controls how an element flows: block, inline, inline-block, or none.
- position controls placement: static (default), relative (offset from normal), absolute (anchored to ancestor), fixed (locked to window), sticky (relative until scrolled).
- z-index decides which positioned elements appear in front.
- float wraps text around an image — modern layouts use Flexbox or Grid instead.
- Flexbox is for one-dimensional layouts — rows of cards, navigation menus, centring content.

- CSS Grid is for two-dimensional layouts — full pages, photo galleries, dashboards.
- @media queries let CSS respond to screen size; mobile-first uses min-width as the default approach.
- Always include <meta name="viewport" content="width=device-width, initial-scale=1.0"> for mobile-friendly pages.
- Common breakpoints: ≤ 600 px mobile, 601-992 px tablet, ≥ 993 px desktop.

10. Practice Questions

A. Short Answer (2 marks each)

1. Differentiate between block, inline, and inline-block display values with one example each.

2. Differentiate between position: relative and position: absolute with code examples.

3. What does z-index do, and which kind of element can use it?

4. What is the main difference between Flexbox and CSS Grid?

5. Write a CSS rule that centres any element horizontally and vertically using Flexbox.

6. What is the role of the <meta name="viewport"> tag in responsive design?

7. Write a media query that switches a 3-column grid to a single column on screens 600 px or narrower.

B. Long Answer (5–10 marks each)

1. Explain the five values of the position property with diagrams and a code example for each.

2. Compare Flexbox and CSS Grid: when to use each, what each is best for, and how they can be used together. Give one full code example for each.

3. Describe at least six Flexbox properties (three on the container, three on items) with examples.

4. Explain what responsive design means and how media queries achieve it. Discuss mobile-first vs desktop-first approaches.

5. Build a responsive 3-column gallery using grid-template-columns: repeat(auto-fit, minmax(...)). Explain how it works.

C. Lab / Hands-On Tasks

1. Build a horizontal navigation menu using Flexbox with five links spaced evenly across the page.

2. Create a photo gallery with six images using CSS Grid (3 columns on desktop, 2 on tablet, 1 on mobile).

3. Make a sticky header that stays at the top of the page when you scroll. Use position: sticky and a high z-index.

4. Design a profile card centred on the screen using display: flex with justify-content and align-items both set to center, and the body height set to 100vh.

5. Take the BCA Department page from Week 5 and convert it to a fully responsive layout: Grid for the page sections, Flexbox for the header, and three media queries for mobile / tablet / desktop.

WEEK 7 — CSS3 EFFECTS & BOOTSTRAP

Course Outcome: CO3 | Topics: Backgrounds, gradients, shadows, transforms, transitions, animations, Bootstrap

Learning Objectives

By the end of this week, students will be able to:

- Style backgrounds with images, gradients, and multiple layers.
- Add depth using box-shadow and text-shadow effects.
- Apply CSS transforms — translate, rotate, scale, and skew.
- Animate elements smoothly using transitions.
- Create multi-step animations using @keyframes.
- Use the Bootstrap framework to build pre-styled, responsive web pages quickly.
- Combine the Bootstrap 12-column grid with components like buttons, cards, and the navbar.

1. Introduction to CSS3 Effects

1.1 What are CSS3 Effects?

CSS3 added many features that used to require image-editing tools or JavaScript: gradients, drop shadows, rounded corners, rotations, smooth hover effects, and full animations. These effects make a web page feel modern, interactive, and polished — without slowing down the browser, because the GPU handles them.

1.2 Why Use Them?

- Faster pages — no image files for borders, shadows, or gradients.
- Crisp on every screen — vector-based, so they look sharp at any size.
- Easy to change — just edit one CSS line, no need to re-export images.
- Better user experience — hover effects, animated buttons, smooth menus.
- Standard support — every modern browser handles them out of the box.

From Week 5 — quick recap

We have already used some basic effects: border-radius for rounded corners, box-shadow for image shadows, and filter for grayscale. This week we go deeper into all the visual tools CSS3 provides.

2. Backgrounds and Gradients

2.1 Background Properties

Example 2.1 — All background properties together

```
.hero {
  background-color: #1a3a5c;              /* solid colour */
  background-image: url('campus.jpg');     /* image */
  background-repeat: no-repeat;            /* no tiling */
  background-size: cover;                 /* fill the box */
  background-position: center;            /* centred */
  background-attachment: fixed;           /* parallax effect */
}
```

Property	What it does
background-color	Solid colour fill — works behind any image.
background-image	url('...') — one or more images stacked.
background-repeat	repeat (default), no-repeat, repeat-x, repeat-y.
background-size	auto, cover (fills box), contain (fits box), or specific px / %.
background-position	center, top right, 50% 50%, 20px 30px, etc.
background-attachment	scroll (default), fixed (parallax), local.
background	Shorthand combining all of the above in one line.

2.2 Linear Gradients

A linear gradient is a smooth blend of two or more colours along a straight line. Use it for hero sections, buttons, banners, or any large coloured area. Gradients count as background-image, so the same properties apply.

Example 2.2 — Different linear gradients

```
/* Default direction (top → bottom) */
.banner1 {
   background: linear-gradient(blue, white);
}

/* Custom direction (left to right) */
.banner2 {
   background: linear-gradient(to right, #1a3a5c, #1f6feb);
}

/* Diagonal — 135 degrees from top-left */
.banner3 {
   background: linear-gradient(135deg, #ff7e5f, #feb47b);
}

/* Three colours with stops */
.banner4 {
   background: linear-gradient(to right,
                        red 0%,
                        yellow 50%,
                        green 100%);
}
```

2.3 Radial and Conic Gradients

Example 2.3 — Radial and conic gradients

```
/* Radial — radiates from the centre outward */
.sun {
  background: radial-gradient(circle, gold, orange, red);
  width: 200px; height: 200px;
  border-radius: 50%;
}

/* Conic — sweeps around like a pie chart */
.pie {
  background: conic-gradient(red 25%, blue 25% 50%, green 50% 75%, gold 75%);
  width: 200px; height: 200px;
  border-radius: 50%;
}
```

3. Shadows

3.1 box-shadow — Drop a Shadow Behind a Box

box-shadow takes up to four lengths and a colour: horizontal-offset, vertical-offset, blur-radius, spread-radius, and the colour. Use it to lift cards, buttons, and modals off the page.

Example 3.1 — Variants of box-shadow

```
/* Subtle card lift */
.card {
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.10);
}

/* Bigger floating shadow */
.modal {
  box-shadow: 0 8px 24px rgba(0, 0, 0, 0.25);
}

/* Inner shadow (inset) — looks like a pressed button */
.pressed {
  box-shadow: inset 0 3px 6px rgba(0, 0, 0, 0.30);
}

/* Multiple shadows separated by commas */
.glow {
  box-shadow: 0 0 10px #1f6feb,
              0 0 20px #1f6feb,
              0 0 30px #1f6feb;
}
```

3.2 text-shadow — Add Glow or Depth to Text

Example 3.2 — Text shadow effects

```
/* Soft drop shadow on a heading */
h1 {
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.4);
}

/* Glowing text */
.neon {
  color: #fff;
  text-shadow: 0 0 8px #1f6feb,
               0 0 16px #1f6feb;
}

/* Embossed look */
.emboss {
  color: #f4f6f8;
  text-shadow: 1px 1px 0 #fff,
              -1px -1px 0 #888;
}
```

4. CSS Transforms

4.1 What is transform?

The transform property lets you move, rotate, resize, or skew an element WITHOUT changing how the page is laid out. It is GPU-accelerated, which means transforms are smooth and fast even on slow devices. They are most powerful when combined with transitions and animations (sections 5 and 6).

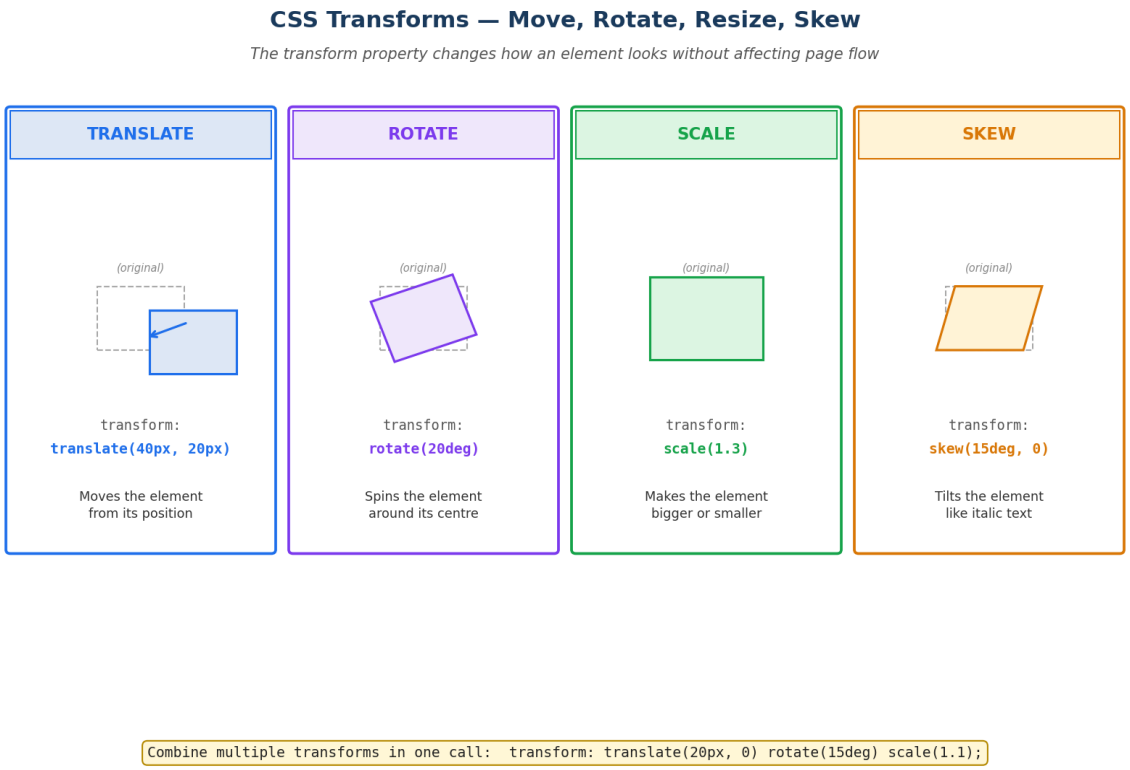


Figure 7.1 — The four most useful transform functions, each shown with the original element as a dashed ghost.

4.2 Transform Functions

Function	What it does
translate(x, y)	Moves the element by x to the right and y down.
translateX(n)	Moves only horizontally.
translateY(n)	Moves only vertically.
rotate(angle)	Rotates around the element's centre. e.g. rotate(45deg).
scale(n)	Resizes uniformly. scale(1.5) = 150% size.
scale(x, y)	Resize horizontally and vertically separately.
skew(x, y)	Tilts the element along the x and y axes.
transform-origin	Sets the pivot point. Default is center center.

4.3 Combining Multiple Transforms

Example 4.1 — Combined transforms and a custom origin

```
/* Apply several transforms at once — order matters! */
.chip {
```

```
    transform: translate(20px, 0)
               rotate(15deg)
               scale(1.1);
}

/* Set the pivot point for rotation */
.clock-hand {
    transform-origin: bottom center;
    transform: rotate(45deg);
}
```

Why use transform instead of margin or position?

transform is much faster — the browser uses the GPU instead of recalculating page layout. It also doesn't push neighbouring elements around. Use transform for visual effects; use margin/position for actual layout.

5. Transitions

5.1 What is a Transition?

A transition makes a CSS property change smoothly over time, instead of instantly. They are usually triggered by a state change like :hover, :focus, or a JavaScript class. Without a transition, hovering a button would make it change colour instantly — with a transition, the colour fades smoothly over a fraction of a second.

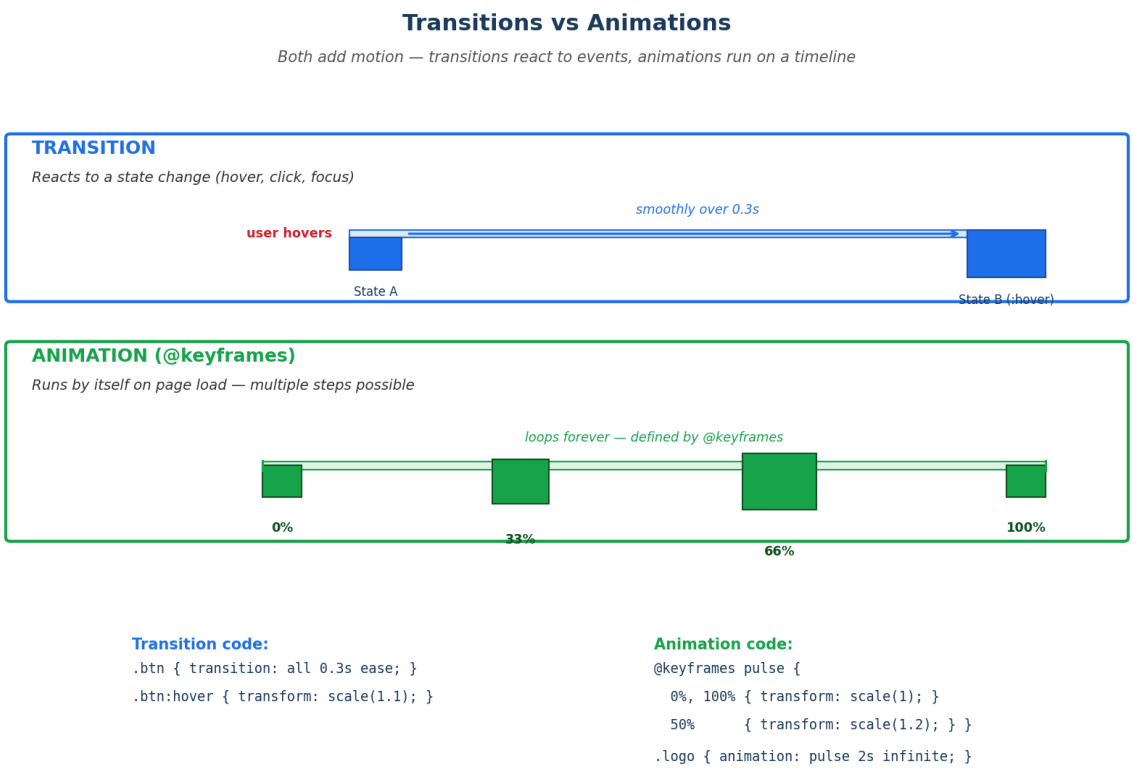


Figure 7.2 — Transitions react to state changes; animations run by themselves on a defined timeline.

5.2 The transition Property

Example 5.1 — Transitions in action

```
/* Basic syntax */
.btn {
   background: #1f6feb;
   transition: background 0.3s ease;
}

.btn:hover {
   background: #16a34a;       /* fades smoothly */
}

/* Animate every property that changes */
.card {
   transition: all 0.4s ease-in-out;
}
.card:hover {
   transform: translateY(-5px);
   box-shadow: 0 8px 16px rgba(0,0,0,0.2);
}

/* Multiple properties with different timings */
.chip {
   transition: background 0.3s ease,
               transform 0.5s ease,
               color 0.2s linear;
}
```

5.3 The Four Parts of a Transition

Part	Meaning
property	Which CSS property to animate. Use 'all' for any property that changes.
duration	How long the change takes — 0.3s, 500ms, 1s.
timing-function	The shape of the change: ease (default), linear, ease-in, ease-out, ease-in-out.
delay	How long to wait before starting — useful for staggered effects.

5.4 A Complete Hover Card Example

Example 5.2 — A card that lifts up smoothly on hover

```
<style>
  .course-card {
    background: white;
    padding: 20px;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.08);
    transition: transform 0.3s ease,
                box-shadow 0.3s ease;
  }
  .course-card:hover {
    transform: translateY(-4px) scale(1.02);
    box-shadow: 0 12px 24px rgba(0,0,0,0.15);
  }
</style>

<div class="course-card">
  <h3>Web Technology</h3>
  <p>BCA 3rd Semester</p>
</div>
```

6. Animations with @keyframes

6.1 Transitions vs Animations

Aspect	Transition	Animation
Trigger	User event (hover, click, focus).	Runs by itself when page loads.
Steps	Only two — start state and end state.	Many — defined as keyframes.
Looping	Runs once per state change.	Can loop forever.
Best for	Hover effects, smooth state changes.	Loaders, banners, attention-grabbing motion.

6.2 Defining @keyframes

An animation has two parts: an @keyframes rule that defines what happens at each step, and an animation property on the element that says when and how to play it.

Example 6.1 — A pulsing logo animation

```
/* Step 1: define the keyframes */
@keyframes pulse {
  0%   { transform: scale(1);   }
  50%  { transform: scale(1.2); }
  100% { transform: scale(1);   }
```

```
}

/* Step 2: apply it to an element */
.logo {
   animation-name: pulse;
   animation-duration: 2s;
   animation-iteration-count: infinite;
   animation-timing-function: ease-in-out;
}

/* Or use the shorthand: name duration timing iterations */
.logo {
   animation: pulse 2s ease-in-out infinite;
}
```

6.3 Common Animation Properties

Property	What it does
animation-name	Which @keyframes rule to use.
animation-duration	How long one cycle lasts.
animation-iteration-count	How many times to play. Use 'infinite' to loop forever.
animation-direction	normal, reverse, alternate (back and forth).
animation-delay	How long to wait before starting.
animation-timing-function	ease, linear, ease-in, ease-out, etc.
animation-fill-mode	What state to keep before/after — forwards, backwards, both.
animation-play-state	running or paused (toggle with JavaScript).

6.4 More Animation Examples

Example 6.2 — Three useful animations

```
/* Slide in from the left */
@keyframes slideIn {
   from { transform: translateX(-100%); opacity: 0; }
   to   { transform: translateX(0);     opacity: 1; }
}
.banner { animation: slideIn 1s ease-out; }

/* Fade in on page load */
@keyframes fadeIn {
   from { opacity: 0; }
```

```
    to   { opacity: 1; }
}
main { animation: fadeIn 0.8s ease; }

/* A simple loading spinner */
@keyframes spin {
    from { transform: rotate(0deg);   }
    to   { transform: rotate(360deg); }
}
.spinner {
    width: 40px; height: 40px;
    border: 4px solid #ddd;
    border-top: 4px solid #1f6feb;
    border-radius: 50%;
    animation: spin 1s linear infinite;
}
```

7. Introduction to Bootstrap

7.1 What is Bootstrap?

Bootstrap is the world's most popular CSS framework — a collection of ready-made styles and components you can drop into any web page. Originally created by Twitter, it lets you build a clean, responsive website in minutes by simply adding class names like btn btn-primary or col-md-6 to your HTML.

7.2 Why Use Bootstrap?

- Saves time — no need to write your own CSS for buttons, navbars, cards, forms, modals.
- Responsive by default — every component adapts to mobile, tablet, and desktop automatically.
- Cross-browser tested — works the same in Chrome, Firefox, Edge, and Safari.
- Huge community — millions of developers, plenty of tutorials and templates online.
- Beginner-friendly — you only need to know HTML and which classes to use.

7.3 Adding Bootstrap to a Page

The easiest way is to add a single <link> tag for the CSS and one <script> tag for JavaScript (only needed if you use interactive components like modals or dropdowns). The version below is from the official Bootstrap CDN — no installation required.

Example 7.1 — A minimal Bootstrap page using the CDN

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>My Bootstrap Page</title>

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
        rel="stylesheet">
</head>
<body>

    <h1 class="text-center text-primary">Hello, Bootstrap!</h1>
    <button class="btn btn-success">Click Me</button>

    <!-- Bootstrap JS (for dropdowns, modals, carousel, etc.) -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
    </script>
</body>
</html>
```

8. The Bootstrap Grid System

8.1 Container, Row, Column

Bootstrap's grid is built on three classes: .container wraps your content, .row groups columns, and .col-* defines each column's width. Every row is split into 12 equal columns. You combine them — col-md-4 means "take 4 of 12 columns on medium screens or larger".

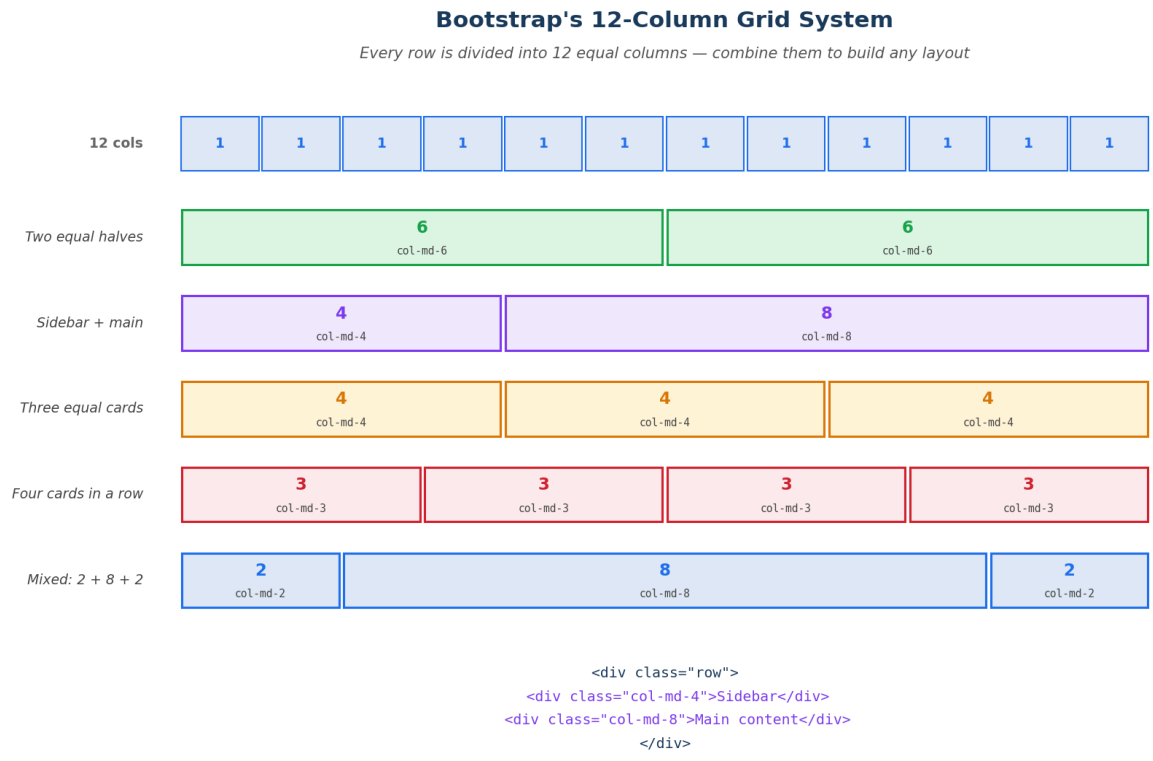


Figure 7.3 — How the 12-column grid divides into common layouts.

8.2 Grid Breakpoints

Class prefix	Applies from	Typical Device
col-	All sizes (extra-small)	Phones
col-sm-	≥ 576 px	Large phones
col-md-	≥ 768 px	Tablets
col-lg-	≥ 992 px	Laptops
col-xl-	≥ 1200 px	Desktops
col-xxl-	≥ 1400 px	Large monitors

8.3 Grid Examples

Example 8.1 — Three grid layouts

```
<div class="container">

  <!-- Two equal halves -->
  <div class="row">
    <div class="col-md-6">Left half</div>
    <div class="col-md-6">Right half</div>
  </div>

  <!-- Sidebar (4) + Main content (8) -->
  <div class="row">
    <div class="col-md-4">Sidebar</div>
    <div class="col-md-8">Main content</div>
  </div>

  <!-- Responsive: 1 col on mobile, 2 on tablet, 3 on desktop -->
  <div class="row">
    <div class="col-12 col-md-6 col-lg-4">Card 1</div>
    <div class="col-12 col-md-6 col-lg-4">Card 2</div>
    <div class="col-12 col-md-6 col-lg-4">Card 3</div>
  </div>

</div>
```

9. Common Bootstrap Components

9.1 Buttons

Example 9.1 — Bootstrap buttons in every flavour

```
<button class="btn btn-primary">Primary</button>
```

```
<button class="btn btn-secondary">Secondary</button>
<button class="btn btn-success">Success</button>
<button class="btn btn-danger">Danger</button>
<button class="btn btn-warning">Warning</button>
<button class="btn btn-info">Info</button>
<button class="btn btn-light">Light</button>
<button class="btn btn-dark">Dark</button>
<button class="btn btn-link">Link</button>

<!-- Outline variants -->
<button class="btn btn-outline-primary">Outline</button>

<!-- Sizes -->
<button class="btn btn-primary btn-lg">Large</button>
<button class="btn btn-primary btn-sm">Small</button>
```

9.2 Cards

Example 9.2 — A card with image, title, text, and button

```
<div class="card" style="width: 18rem;">
  <img src="campus.jpg" class="card-img-top" alt="Campus">
  <div class="card-body">
    <h5 class="card-title">BCA Programme</h5>
    <p class="card-text">Three-year degree at RVSCET, Jamshedpur.</p>
    <a href="#" class="btn btn-primary">Apply Now</a>
  </div>
</div>
```

9.3 Navbar

Example 9.3 — A responsive navbar (collapses to a hamburger menu on mobile)

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">RVSCET</a>
    <button class="navbar-toggler" type="button"
            data-bs-toggle="collapse" data-bs-target="#nav">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="nav">
      <ul class="navbar-nav">
        <li class="nav-item"><a class="nav-link" href="#">Home</a></li>
        <li class="nav-item"><a class="nav-link" href="#">About</a></li>
        <li class="nav-item"><a class="nav-link" href="#">Contact</a></li>
      </ul>
    </div>
  </div>
</nav>
```

9.4 Forms

Example 9.4 — A clean Bootstrap login form

```
<form>
  <div class="mb-3">
    <label for="email" class="form-label">Email</label>
    <input type="email" class="form-control" id="email">
  </div>
  <div class="mb-3">
    <label for="pwd" class="form-label">Password</label>
    <input type="password" class="form-control" id="pwd">
  </div>
  <button type="submit" class="btn btn-primary">Login</button>
</form>
```

9.5 Alerts

Example 9.5 — Coloured alert boxes

```
<div class="alert alert-success">Registration successful!</div>
<div class="alert alert-danger">Invalid email or password.</div>
<div class="alert alert-warning">Your session will expire in 5 minutes.</div>
<div class="alert alert-info">Bootstrap 5 is now available.</div>
```

9.6 Utility Classes — Quick Wins

Class	What it does
text-center, text-end	Align text horizontally.
text-primary, text-danger	Apply theme colours to text.
bg-primary, bg-light	Apply background colours.
m-3, p-3	Margin / padding (1 to 5, on all sides).
mt-2, mb-4, mx-auto	Margin top / bottom / horizontal auto.
d-flex, d-none	Display flex / hidden.
shadow, rounded	Quick shadows and rounded corners.
w-100, h-50	Width and height as a percentage.

10. Worked Example — A Complete Bootstrap Page

This example combines everything: a navbar, a hero with a gradient background, a 3-card grid, hover effects on the cards using a small custom CSS, and a footer.

Example 10.1 — A complete Bootstrap page with hero, cards, hover transitions, and a fade-in animation

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>BCA at RVSCET</title>
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
      rel="stylesheet">
  <style>
    .hero {
      background: linear-gradient(135deg, #1a3a5c, #1f6feb);
      color: white;
      padding: 80px 20px;
      text-align: center;
    }
    .course-card {
      transition: transform 0.3s, box-shadow 0.3s;
    }
    .course-card:hover {
      transform: translateY(-6px);
      box-shadow: 0 12px 24px rgba(0,0,0,0.15);
    }
    @keyframes fadeIn {
      from { opacity: 0; transform: translateY(20px); }
      to   { opacity: 1; transform: translateY(0);    }
    }
    .hero h1 { animation: fadeIn 0.8s ease; }
  </style>
</head>
<body>

  <!-- Navbar -->
  <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class="container">
      <a class="navbar-brand" href="#">RVSCET — BCA</a>
      <ul class="navbar-nav ms-auto">
        <li class="nav-item"><a class="nav-link" href="#">Home</a></li>
        <li class="nav-item"><a class="nav-link" href="#">Courses</a></li>
        <li class="nav-item"><a class="nav-link" href="#">Contact</a></li>
      </ul>
    </div>
  </nav>

  <!-- Hero -->
  <header class="hero">
    <h1 class="display-4 fw-bold">Bachelor of Computer Applications</h1>
    <p class="lead">A three-year programme at RVSCET, Jamshedpur</p>
    <a href="#" class="btn btn-warning btn-lg">Apply for 2026</a>
```

```
  </header>

  <!-- Three course cards -->
  <main class="container my-5">
    <h2 class="text-center mb-4">Core Subjects</h2>
    <div class="row g-4">

      <div class="col-12 col-md-6 col-lg-4">
        <div class="card course-card h-100">
          <div class="card-body">
            <h5 class="card-title text-primary">Web Technology</h5>
            <p class="card-text">HTML5, CSS3, JavaScript, PHP and beyond.</p>
          </div>
        </div>
      </div>

      <div class="col-12 col-md-6 col-lg-4">
        <div class="card course-card h-100">
          <div class="card-body">
            <h5 class="card-title text-primary">Data Structures</h5>
            <p class="card-text">Arrays, lists, trees, graphs, and algorithms.</p>
          </div>
        </div>
      </div>

      <div class="col-12 col-md-6 col-lg-4">
        <div class="card course-card h-100">
          <div class="card-body">
            <h5 class="card-title text-primary">Database Systems</h5>
            <p class="card-text">MySQL, queries, joins, and relational design.</p>
          </div>
        </div>
      </div>

    </div>
  </main>

  <!-- Footer -->
  <footer class="bg-dark text-light text-center py-3">
    <p class="mb-0">&copy; 2026 RVSCET, Jamshedpur</p>
  </footer>

  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js">
  </script>
</body>
</html>
```

Try this in your lab

1. Save the code above as bca.html.
2. Open it in Chrome — Bootstrap loads from the internet via the CDN.
3. Resize the browser from wide to narrow — the navbar collapses, cards stack.
4. Hover over the cards — they should lift up smoothly.
5. Reload the page — the hero heading fades in.

11. Summary of Week 7

- Backgrounds support solid colours, images, and gradients (linear, radial, conic).
- box-shadow and text-shadow add depth without using image files.
- transform: translate / rotate / scale / skew — fast, GPU-accelerated visual effects.
- transition smoothly animates property changes on state events like :hover.
- @keyframes + animation define multi-step, looping animations that run on their own.
- Bootstrap is a CSS framework with ready-made styles and responsive components.
- Bootstrap's grid uses .container > .row > .col-* with 12 columns per row and breakpoints sm/md/lg/xl/xxl.
- Common Bootstrap components include buttons, cards, navbar, forms, alerts, plus utility classes for margin, padding, alignment, and colour.
- You can mix Bootstrap with your own custom CSS and animations for the best of both worlds.

12. Practice Questions

A. Short Answer (2 marks each)

1. Write CSS that gives a div a linear gradient background from blue to white going left to right.

2. What is the difference between transition and animation in CSS?

3. Write the CSS to scale an element to 1.2x its size only when the user hovers over it (smoothly over 0.3s).

4. Define @keyframes spin that rotates an element from 0 to 360 degrees.

5. What is Bootstrap and why do developers use it?

6. How many columns does the Bootstrap grid use per row?

7. Write the HTML for a Bootstrap row with a 4-column sidebar and an 8-column main area on medium screens.

B. Long Answer (5–10 marks each)

1. Explain box-shadow and text-shadow with code examples for at least three different visual effects each.

2. Describe the four main transform functions (translate, rotate, scale, skew) with a code example and a use case for each.

3. Compare CSS transitions and animations: trigger, number of steps, looping, and best use cases. Provide one full example of each.

4. Explain Bootstrap's grid system in detail: container, row, columns, breakpoints, and the 12-column rule. Give a code example showing a layout that changes from 1 column on mobile to 3 columns on desktop.

5. List and describe at least six common Bootstrap components with HTML code for each.

C. Lab / Hands-On Tasks

1. Create a button that uses a linear gradient background, has rounded corners and a shadow, and smoothly lifts up (translateY) on hover with a transition.

2. Create a loading spinner using @keyframes spin and a circular div with one coloured side of the border.

3. Build a hover card that, when hovered, scales up to 1.05 and shows a stronger shadow — both animated smoothly.

4. Build a complete responsive landing page for the BCA Department using Bootstrap. Include a navbar, a hero section with a gradient background, three cards in a row (one column on mobile, three on desktop), and a footer.

5. Take any earlier lab page (e.g. the Week 4 registration form) and re-style it entirely with Bootstrap form classes (form-control, form-label, btn, alert).

WEEK 8 — JavaScript Basics

Course Outcome: CO3 **Topics:** What is JavaScript, syntax, variables, data types, operators, control flow, functions, output methods

---

Learning Objectives

By the end of this week, students will be able to:

- Explain what JavaScript is and where it runs.
- Add JavaScript to a web page in three different ways.
- Declare variables using *let*, *const*, and *var* correctly.
- Identify and use JavaScript's main data types.
- Apply arithmetic, comparison, logical, and assignment operators.
- Write conditional statements using *if*, *else if*, *else*, and *switch*.
- Use *for*, *while*, and *do-while* loops.
- Define and call functions, including arrow functions.
- Display output using *alert()*, *console.log()*, *document.write()*, and *innerHTML*.

---

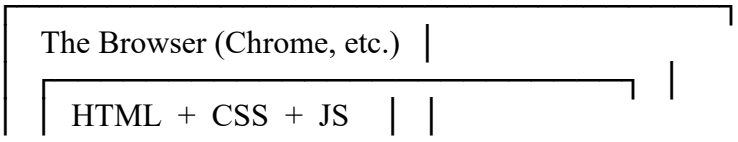
1. Introduction to JavaScript

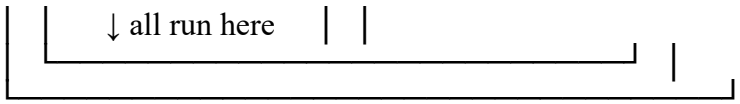
1.1 What is JavaScript?

JavaScript (JS) is the **programming language of the web**. It runs inside every web browser and is what makes web pages **interactive** — clickable buttons, form validation, image sliders, animations, games, and full single-page applications. While HTML provides the structure and CSS provides the look, JavaScript provides the **behaviour**.

> **Quick analogy** If a web page were a person: - **HTML** is the skeleton (structure) - **CSS** is the clothes and skin (appearance) - **JavaScript** is the muscles and brain (movement and decisions)

1.2 Where Does JavaScript Run?





JavaScript started as a browser-only language. Today, with **Node.js**, the same language also runs on servers, on mobile apps (React Native), and on desktop apps (Electron). For this course, we focus on JavaScript in the browser.

1.3 A Brief History

Year	Milestone
1995	Created by Brendan Eich at Netscape in just 10 days.
1997	Standardised as **ECMAScript (ES)** — the official name.
2009	Node.js launched — JS leaves the browser.
2015	**ES6 (ES2015)** — major upgrade: *let*, *const*, arrow functions, classes.
2016 onwards	New features added every year (ES2016, ES2017, …).

> **Note:** "JavaScript" and "Java" are completely different languages. The name is a marketing legacy from the 1990s.

2. Adding JavaScript to a Web Page

There are **three** ways to include JavaScript in your HTML — exactly mirroring the three ways to include CSS that we learned in Week 5.

2.1 Inline JavaScript — using event attributes

```
<button onclick="alert('Hello, RVSCET!')">Click Me</button>
```

> **Output in browser:** clicking the button shows a popup saying *Hello, RVSCET!*

2.2 Internal JavaScript — using *<script>*

```
<!DOCTYPE html>
<html>
<head>
  <title>Internal JS Demo</title>
</head>
<body>
  <h1 id="greeting">Hello!</h1>

  <script>
    document.getElementById("greeting").innerHTML = "Welcome to BCA";
    console.log("Script ran successfully.");
  </script>
</body>
</html>
```

2.3 External JavaScript — using a separate *.js* file

This is the **recommended approach** for any real project.

```
<!-- File: index.html -->
<!DOCTYPE html>
<html>
<head>
  <title>External JS Demo</title>
</head>
<body>
  <h1 id="greeting">Hello!</h1>

  <!-- Best practice: place script tag at the end of body -->
  <script src="app.js"></script>
</body>
</html>
```

```
// File: app.js
document.getElementById("greeting").innerHTML = "Welcome to BCA";
console.log("Script ran successfully.");
```

2.4 Where to Place *<script>*

Location	Effect
Inside *<head>*	Loads early — but blocks page rendering.
Just before *</body>*	**Recommended.** HTML loads first, then the script runs.
<script defer src="...">	Modern alternative — loads in background, runs after HTML.

Tip — Comments

```
// This is a single-line comment
/* This is a multi-line
   comment */
```

3. Variables and Constants

A **variable** is a named storage box that holds a value. JavaScript has three keywords for declaring them: *let*, *const*, and *var*.

3.1 Declaring Variables

```
let studentName = "Aman Kumar";      // can be changed later
const collegeName = "RVSCET";        // cannot be changed
var rollNumber = 101;                // old style — avoid in new code

console.log(studentName);            // Aman Kumar
console.log(collegeName);            // RVSCET
console.log(rollNumber);             // 101
```

3.2 *let* vs *const* vs *var*

Keyword	Reassignable?	Block-scoped?	Use when…
let	Yes	Yes	The value will change later.
const	No	Yes	The value is fixed for the program. **Default choice.**
var	Yes	No (function-scoped)	Avoid in new code — kept for legacy support.

Modern rule of thumb: use *const* by default. If you find that you need to change the value, switch to *let*. Almost never use *var*.

3.3 Naming Rules

- Must begin with a letter, _, or *$*. Cannot start with a digit.
- Case-sensitive: *name* and *Name* are different.
- Cannot be a reserved keyword (*if*, *class*, *return*, …).
- Convention: **camelCase** — *firstName*, *totalMarks*, *isLoggedIn*.

```
// Valid
let firstName = "Aman";
let _temp = 42;
let $price = 199;

// Invalid
// let 1stName = "Aman";     ✗ starts with a digit
// let first-name = "Aman";  ✗ contains a hyphen
// let class = "BCA";        ✗ reserved word
```

4. Data Types

JavaScript values fall into two groups: **primitive** types and **objects**.

4.1 Primitive Types

```
let name    = "Riya Singh";      // String
let age     = 21;                // Number
let height  = 5.6;               // Number (decimals are also Number)
let isAdult = true;              // Boolean
let address = null;              // Null  (intentional empty value)
let phone;                       // Undefined (no value assigned)
```

Type	Description	Examples
String	Text, in quotes.	*"Aman"*, *'BCA'*, *`hello`*
Number	Whole or decimal numbers.	*42*, *3.14*, *-7*
Boolean	*true* or *false*.	*true*, *false*
Null	Intentionally “no value”.	*null*
Undefined	Variable declared but not assigned.	*undefined*

Type	Description	Examples
BigInt	Very large integers.	*9007199254740993n*
Symbol	Unique identifier (advanced).	*Symbol("id")*

4.2 Object Types

```
// Object — collection of key-value pairs
let student = {
   name: "Aman",
   age: 21,
   course: "BCA"
};

// Array — ordered list of values
let subjects = ["Web Tech", "DSA", "Maths"];

// Function — reusable block of code (also a value!)
let greet = function() { console.log("Hello!"); };
```

4.3 Checking the Type — *typeof*

```
console.log(typeof "Hello");     // "string"
console.log(typeof 42);          // "number"
console.log(typeof true);        // "boolean"
console.log(typeof undefined);   // "undefined"
console.log(typeof null);        // "object"  ← historical quirk!
console.log(typeof [1, 2, 3]);   // "object"
console.log(typeof {a: 1});      // "object"
```

4.4 Strings in Detail

```
let a = "Aman";
let b = 'Riya';
let c = `Vivek`;    // backticks → template literal

// Concatenation
let greeting = "Hello, " + a + "!";      // Hello, Aman!

// Template literals — much cleaner
let age = 21;
let info = `My name is ${a} and I am ${age} years old.`;

// Useful string methods
let msg = "  Welcome to BCA  ";
console.log(msg.length);         // 19
console.log(msg.trim());         // "Welcome to BCA"
console.log(msg.toUpperCase());  // "  WELCOME TO BCA  "
console.log(msg.toLowerCase());  // "  welcome to bca  "
console.log(msg.includes("BCA")); // true
console.log(msg.replace("BCA", "MCA"));  // "  Welcome to MCA  "
```

4.5 Numbers in Detail

```
let a = 10;
let b = 3.14;
let c = 0.1 + 0.2;          // 0.30000000000000004 (floating-point quirk)

// Useful Math object methods
console.log(Math.PI);        // 3.141592653589793
console.log(Math.round(4.6)); // 5
console.log(Math.floor(4.9)); // 4
console.log(Math.ceil(4.1));  // 5
console.log(Math.max(2, 5, 1)); // 5
console.log(Math.min(2, 5, 1)); // 1
console.log(Math.random());     // a number between 0 and 1

// Parse strings into numbers
let n1 = Number("42");       // 42
let n2 = parseInt("42 marks"); // 42
let n3 = parseFloat("3.14em"); // 3.14
```

5. Operators

5.1 Arithmetic Operators

```
let a = 10, b = 3;

console.log(a + b);  // 13   addition
console.log(a - b);  // 7    subtraction
console.log(a * b);  // 30   multiplication
console.log(a / b);  // 3.333...   division
console.log(a % b);  // 1    modulus (remainder)
console.log(a ** b); // 1000   exponent (a to the power b)

// Increment / decrement
let x = 5;
x++;              // x is now 6
x--;              // x is now 5
```

5.2 Assignment Operators

```
let n = 10;

n += 5;   // n = n + 5  → 15
n -= 3;   // n = n - 3  → 12
n *= 2;   // n = n * 2  → 24
n /= 4;   // n = n / 4  → 6
n %= 4;   // n = n % 4  → 2
```

5.3 Comparison Operators

```
console.log(5 == "5");   // true   (loose equality — converts types)
console.log(5 === "5");  // false  (strict equality — types must match)
```

```
console.log(5 != "5");    // false
console.log(5 !== "5");   // true
console.log(5 > 3);       // true
console.log(5 < 3);       // false
console.log(5 >= 5);      // true
console.log(5 <= 4);      // false
```

> **Important:** always prefer === and !== (strict equality). They prevent surprising bugs from automatic type conversion.

5.4 Logical Operators

```
let age = 20;
let hasID = true;

console.log(age >= 18 && hasID);   // true   AND
console.log(age >= 18 || hasID);   // true   OR
console.log(!hasID);               // false  NOT
```

Operator	Name	Result
&&	AND	true only if **both** sides are true.
||	OR	true if **at least one** side is true.
!	NOT	flips true to false and vice versa.

5.5 String Concatenation with +

```
let firstName = "Aman";
let lastName = "Kumar";
let full = firstName + " " + lastName;     // "Aman Kumar"

// Watch out — + with strings concatenates, not adds!
console.log("5" + 3);        // "53"  (string)
console.log(5 + 3);          // 8     (number)
console.log("Total: " + 10); // "Total: 10"
```

6. Control Flow

6.1 The *if* Statement

```
let marks = 75;

if (marks >= 40) {
   console.log("Pass");
}
```

6.2 *if … else*

```
let age = 17;

if (age >= 18) {
   console.log("You can vote.");
```

```
} else {
   console.log("You are not yet eligible to vote.");
}
```

6.3 *if … else if … else*

```
let marks = 78;
let grade;

if (marks >= 90)      grade = "A+";
else if (marks >= 80) grade = "A";
else if (marks >= 70) grade = "B";
else if (marks >= 60) grade = "C";
else if (marks >= 40) grade = "D";
else                  grade = "F";

console.log(`Your grade is ${grade}`);   // "Your grade is B"
```

6.4 The Ternary Operator

A short, one-line form of *if-else*:

```
let age = 20;
let status = (age >= 18) ? "Adult" : "Minor";
console.log(status);     // Adult
```

6.5 The *switch* Statement

Useful when checking one value against many possibilities.

```
let day = 3;
let dayName;

switch (day) {
   case 1: dayName = "Monday";    break;
   case 2: dayName = "Tuesday";   break;
   case 3: dayName = "Wednesday"; break;
   case 4: dayName = "Thursday";  break;
   case 5: dayName = "Friday";    break;
   default: dayName = "Weekend";
}

console.log(dayName);    // Wednesday
```

Don't forget *break*! Without it, the program "falls through" to the next case.

7. Loops

7.1 The *for* Loop

```
//      start;     condition;  step
for (let i = 1; i <= 5; i++) {
   console.log("Hello " + i);
}
```

```
// Output:
// Hello 1
// Hello 2
// Hello 3
// Hello 4
// Hello 5
```

Parts of a *for* loop:

```
for ( INIT ; CONDITION ; STEP ) {
      |        |          |
      |        |          └── runs after every iteration
      |        └── checked before each iteration
      └── runs once at the start
}
```

7.2 The *while* Loop

Used when you don't know in advance how many iterations are needed.

```
let n = 1;

while (n <= 5) {
   console.log("n is " + n);
   n++;
}
```

7.3 The *do…while* Loop

The body runs **at least once**, even if the condition is false from the start.

```
let i = 10;

do {
   console.log("Value: " + i);
   i++;
} while (i <= 5);
// Output: Value: 10  (runs once even though 10 > 5)
```

7.4 *break* and *continue*

```
// break — exit the loop entirely
for (let i = 1; i <= 10; i++) {
   if (i === 5) break;
   console.log(i);
}
// Output: 1 2 3 4

// continue — skip just this iteration
for (let i = 1; i <= 5; i++) {
   if (i === 3) continue;
   console.log(i);
}
// Output: 1 2 4 5
```

7.5 Looping Through an Array

```
let subjects = ["Web Tech", "DSA", "Maths", "English"];

// Classic for loop
for (let i = 0; i < subjects.length; i++) {
  console.log(subjects[i]);
}

// for...of — modern, cleaner
for (let subject of subjects) {
  console.log(subject);
}
```

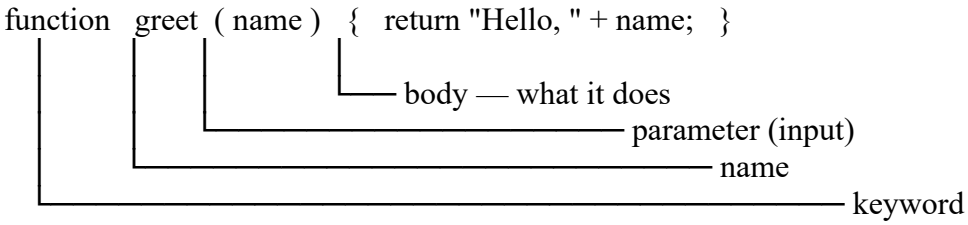
8. Functions

A **function** is a reusable block of code that performs a task. Define it once, call it many times.

8.1 Function Declaration

```
function greet(name) {
  return "Hello, " + name + "!";
}

console.log(greet("Aman"));   // Hello, Aman!
console.log(greet("Riya"));   // Hello, Riya!
```

8.2 Parts of a Function



8.3 Function Expression

A function stored in a variable:

```
const square = function(n) {
  return n * n;
};

console.log(square(5));   // 25
```

8.4 Arrow Functions (ES6)

Shorter syntax for function expressions — common in modern JavaScript:

```
// Standard form
const add = (a, b) => {
  return a + b;
};

// One-line form (implicit return)
```

```
const multiply = (a, b) => a * b;

// Single parameter — parentheses optional
const square = n => n * n;

// No parameters
const sayHi = () => console.log("Hi!");

console.log(add(3, 4));       // 7
console.log(multiply(5, 6));  // 30
console.log(square(7));       // 49
sayHi();                      // Hi!
```

8.5 Default Parameters

```
function greet(name = "Guest") {
  return "Welcome, " + name;
}

console.log(greet("Aman"));   // Welcome, Aman
console.log(greet());         // Welcome, Guest
```

8.6 Functions Calling Functions

```
function area(width, height) {
  return width * height;
}

function totalArea(rooms) {
  let total = 0;
  for (let r of rooms) {
    total += area(r.width, r.height);
  }
  return total;
}

let myRooms = [
  { width: 10, height: 12 },
  { width:  8, height: 10 },
  { width: 14, height: 15 }
];

console.log(totalArea(myRooms));   // 410
```

---

9. Output Methods

JavaScript has four common ways to display data:

Method	Where it Shows	Best Used For
alert("text")	Popup dialog box.	Quick messages and warnings.
console.log("text")	Browser DevTools console.	Debugging.

Method	Where it Shows	Best Used For
document.write("text")	Replaces the entire page.	Demos only — avoid in real sites.
element.innerHTML = "..."	Inside a specific HTML element.	Updating part of the page.

9.1 Examples of Each

```
<!DOCTYPE html>
<html>
<body>

  <h1 id="title">Hello!</h1>
  <p id="message"></p>

  <button onclick="showAlert()">Show Alert</button>

  <script>
    // 1. alert
    function showAlert() {
      alert("This is an alert popup.");
    }

    // 2. console.log
    console.log("This appears in DevTools (F12 → Console)");

    // 3. innerHTML — best for updating the page
    document.getElementById("title").innerHTML = "Welcome to BCA";
    document.getElementById("message").innerHTML =
      "JavaScript can change content dynamically!";
  </script>

</body>
</html>
```

> **Tip — Chrome DevTools** Press **F12** (or right-click → Inspect) and switch to the **Console** tab to see the output of *console.log()*. This is the most-used debugging tool for JavaScript.

---

10. Worked Example — Marks Grading System

Let's combine everything from this week. We'll build a small page that asks for a student's marks and displays a grade with a personalised message.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Marks Grading System</title>
  <style>
    body { font-family: Arial; padding: 20px; max-width: 500px; }
```

```
    input, button { padding: 8px; margin: 5px 0; font-size: 16px; }
    #result { margin-top: 15px; padding: 10px;
              background: #f0f8ff; border-radius: 6px; }
  </style>
</head>
<body>

  <h1>Student Grading System</h1>

  <label>Student name:</label><br>
  <input type="text" id="nameInput"><br>

  <label>Marks (0-100):</label><br>
  <input type="number" id="marksInput" min="0" max="100"><br>

  <button onclick="calculateGrade()">Calculate Grade</button>

  <div id="result"></div>

  <script>
    function getGrade(marks) {
      if (marks >= 90) return "A+";
      else if (marks >= 80) return "A";
      else if (marks >= 70) return "B";
      else if (marks >= 60) return "C";
      else if (marks >= 40) return "D";
      else return "F";
    }

    function getMessage(grade) {
      switch (grade) {
        case "A+": return "Outstanding!";
        case "A":  return "Excellent work.";
        case "B":  return "Good job.";
        case "C":  return "Keep working hard.";
        case "D":  return "You passed — but aim higher.";
        case "F":  return "Don't give up. Try again next time.";
        default:   return "";
      }
    }

    function calculateGrade() {
      const name  = document.getElementById("nameInput").value;
      const marks = Number(document.getElementById("marksInput").value);

      if (name === "" || isNaN(marks)) {
        document.getElementById("result").innerHTML =
          "Please enter a valid name and marks.";
        return;
      }
```

```
        const grade = getGrade(marks);
        const msg   = getMessage(grade);

        document.getElementById("result").innerHTML =
          `<strong>${name}</strong>, you scored ${marks}/100.<br>
           Grade: <strong>${grade}</strong> — ${msg}`;
    }
  </script>

</body>
</html>
```

Try this in your lab 1. Save as *grade.html*. 2. Open in Chrome. 3. Enter "Aman" and 78 — click Calculate Grade. You should see grade B and "Good job." 4. Open DevTools (F12) and try *getGrade(95)* directly in the Console — you should see *"A+"*.

11. Summary

- JavaScript is the language that makes web pages interactive — runs in every browser.
- Add JS via inline event attributes, internal *<script>* blocks, or external *.js* files.
- Use *const* by default, *let* if the value will change, and avoid *var* in new code.
- Primitive data types: String, Number, Boolean, Null, Undefined.
- Use *===* and *!==* for safer comparisons (avoid type coercion).
- Control flow uses *if / else if / else*, the ternary operator, and *switch*.
- Loops include *for*, *while*, *do…while*, plus *break* and *continue*.
- Functions are reusable blocks — declare with *function* or as arrow functions *(a, b) => a + b*.
- Output methods: *alert()*, *console.log()*, *document.write()*, and *innerHTML*.

12. Practice Questions

A. Short Answer (2 marks each)

1. What is the difference between *let*, *const*, and *var*?
2. List any five primitive data types in JavaScript.
3. Explain the difference between == and ===.
4. Write a single line that uses a template literal to print "Welcome, Aman" given *let name = "Aman"*.
5. What is the difference between *while* and *do…while* loops?
6. Convert the following function into an arrow function: *function double(n) { return n * 2; }*
7. Where do you see the output of *console.log()*?

B. Long Answer (5–10 marks each)

1. Explain the three ways to add JavaScript to an HTML page with one example for each. Which one is the best practice and why?
2. Describe all primitive data types of JavaScript with a code example for each, and explain the result of *typeof null*.
3. Describe the arithmetic, comparison, and logical operators of JavaScript with at least two examples each.

4. Explain the *if-else if-else*, ternary operator, and *switch* statement with a separate example for each.
5. Explain the three loops in JavaScript (*for*, *while*, *do…while*) with examples, and discuss when each one is appropriate.

C. Lab / Hands-On Tasks

1. Write a JavaScript program that takes a year as input and prints whether it is a leap year. (Hint: a year is a leap year if it is divisible by 4 but not 100, OR if it is divisible by 400.)
2. Write a function *isPrime(n)* that returns *true* if *n* is a prime number, *false* otherwise. Test it for 1, 2, 7, 12, 17, 25.
3. Use a *for* loop to print the multiplication table of any number entered through *prompt()*.
4. Build a simple calculator page with two number inputs, a dropdown of operations (+, -, ×, ÷), and a button that displays the result on the page.
5. Take the marks-grading example from this chapter and extend it: also accept the student's class and total subjects, and calculate average marks for at least three subjects.

WEEK 9 — JavaScript Advanced (DOM, Events, JSON)

Course Outcome: CO3 **Topics:** Arrays, objects, the DOM, selecting and modifying elements, events, event listeners, form validation, JSON

Learning Objectives

By the end of this week, students will be able to:

- Create and manipulate arrays and objects in JavaScript.
- Explain what the DOM is and how the browser builds it.
- Select HTML elements using *getElementById*, *getElementsByClassName*, and *querySelector*.
- Modify text, HTML content, attributes, and CSS styles dynamically.
- Add and remove elements from the page using JavaScript.
- Handle user events with *onclick* and *addEventListener*.
- Validate form inputs using JavaScript.
- Read, write, and parse JSON data.

1. Arrays

An **array** is an ordered list of values, all stored under one name. Items are accessed by their index (position), starting from **0**.

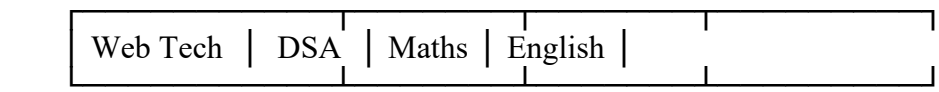
1.1 Creating Arrays

```
// Empty array
let empty = [];

// Array of strings
let subjects = ["Web Tech", "DSA", "Maths", "English"];

// Array of numbers
let marks = [78, 92, 85, 67, 90];
```

```
// Mixed types are allowed (but usually not recommended)
let mixed = [1, "two", true, null, [3, 4]];
```

1.2 Accessing Elements

```
let subjects = ["Web Tech", "DSA", "Maths", "English"];

console.log(subjects[0]);          // "Web Tech"
console.log(subjects[2]);          // "Maths"
console.log(subjects.length);      // 4
console.log(subjects[subjects.length - 1]);  // "English" — last element
```



1.3 Modifying Arrays

```
let nums = [10, 20, 30];

// Change an element
nums[1] = 25;            // [10, 25, 30]

// Add to the end
nums.push(40);           // [10, 25, 30, 40]

// Remove from the end
nums.pop();              // [10, 25, 30] — pop returns 40

// Add to the beginning
nums.unshift(5);         // [5, 10, 25, 30]

// Remove from the beginning
nums.shift();            // [10, 25, 30] — shift returns 5

// Find the position
console.log(nums.indexOf(25));    // 1
console.log(nums.includes(99));   // false
```

1.4 Useful Array Methods

```
let marks = [78, 92, 85, 67, 90];

// Sort (ascending)
console.log([...marks].sort((a, b) => a - b));   // [67, 78, 85, 90, 92]

// Reverse
console.log([...marks].reverse());               // [90, 67, 85, 92, 78]

// Sum using reduce
let total = marks.reduce((sum, m) => sum + m, 0);
```

```
console.log(total);                    // 412

// Average
let avg = total / marks.length;
console.log(avg);                      // 82.4

// Filter — keep only items that pass a test
let passing = marks.filter(m => m >= 80);
console.log(passing);                  // [92, 85, 90]

// Map — transform every item
let percentages = marks.map(m => m + "%");
console.log(percentages);              // ["78%", "92%", ...]

// forEach — run a function on every item
marks.forEach((m, i) => {
  console.log(`Subject ${i + 1}: ${m} marks`);
});
```

Method	What it does
push(x) / pop()	Add / remove from the **end**.
unshift(x) / shift()	Add / remove from the **beginning**.
indexOf(x) / includes(x)	Find an item / check if it exists.
sort() / reverse()	Sort or reverse the array.
slice(a, b)	Return a copy of items from index *a* to *b-1*.
splice(i, n)	Remove (and optionally insert) at position *i*.
forEach(fn)	Run a function on every item.
map(fn)	Build a new array by transforming every item.
filter(fn)	Build a new array of items that pass a test.
reduce(fn, init)	Combine all items into one value (sum, max, …).
join("-")	Convert array into a string with a separator.

2. Objects

An **object** holds related data as **key-value pairs** — like a record card. Use objects when the data has named fields rather than just a sequence.

2.1 Creating an Object

```
let student = {
  name: "Aman Kumar",
  rollNo: 101,
  course: "BCA",
  semester: 3,
  isHosteler: true
};
```

2.2 Accessing Properties

```
// Dot notation (most common)
console.log(student.name);        // "Aman Kumar"
console.log(student.rollNo);      // 101

// Bracket notation (when the key is in a variable)
let key = "course";
console.log(student[key]);        // "BCA"
```

2.3 Modifying Objects

```
// Change a value
student.semester = 4;

// Add a new property
student.email = "aman@rvscet.com";

// Remove a property
delete student.isHosteler;

console.log(student);
```

2.4 Methods Inside Objects

A function stored inside an object is called a **method**.

```
let calculator = {
  a: 10,
  b: 5,
  add: function() {
    return this.a + this.b;
  },
  subtract() {                  // shorter ES6 syntax
    return this.a - this.b;
  }
};

console.log(calculator.add());      // 15
console.log(calculator.subtract());  // 5
```

this inside a method refers to the object itself.

2.5 Looping Through an Object

```
let student = { name: "Aman", roll: 101, course: "BCA" };

for (let key in student) {
  console.log(`${key}: ${student[key]}`);
}
// Output:
// name: Aman
// roll: 101
// course: BCA
```

```
// Get just the keys, values, or both
console.log(Object.keys(student));     // ["name", "roll", "course"]
console.log(Object.values(student));   // ["Aman", 101, "BCA"]
console.log(Object.entries(student));  // [["name","Aman"], ["roll",101], ...]
```

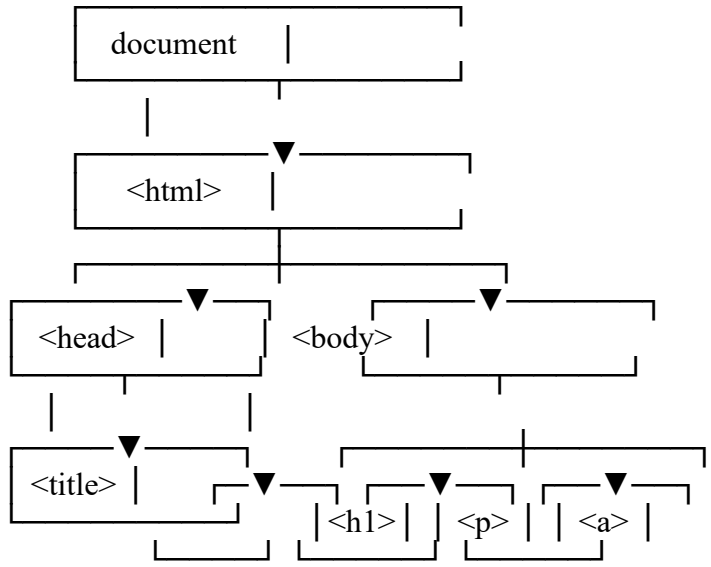
2.6 Array of Objects (very common pattern)

```
let students = [
    { name: "Aman",  marks: 78 },
    { name: "Riya",  marks: 92 },
    { name: "Vivek", marks: 67 }
];

// Find top scorer
let top = students.reduce((best, s) =>
    s.marks > best.marks ? s : best
);
console.log(top);          // { name: "Riya", marks: 92 }

// Print everyone's name and marks
students.forEach(s => {
    console.log(`${s.name} scored ${s.marks}`);
});
```

3. The Document Object Model (DOM)

3.1 What is the DOM?

When the browser loads an HTML page, it builds a **tree of objects** in memory representing every tag, attribute, and piece of text. This tree is called the **Document Object Model**, or **DOM** for short. JavaScript reads and changes this tree to make pages interactive.



3.2 The *document* Object

document is JavaScript's gateway into the DOM. From it, we can reach every element on the page.

```
console.log(document.title);          // current page title
console.log(document.URL);            // page URL
console.log(document.body);           // the <body> element

document.title = "My New Title";      // change the title
document.body.style.background = "lightblue"; // change the background
```

---

4. Selecting Elements

4.1 Five Common Methods

```
// 1. By id (returns one element or null)
let title = document.getElementById("main-title");

// 2. By class name (returns a live HTMLCollection)
let cards = document.getElementsByClassName("card");

// 3. By tag name
let allParagraphs = document.getElementsByTagName("p");

// 4. querySelector — uses CSS selectors, returns the FIRST match
let firstBtn = document.querySelector(".btn");
let logo     = document.querySelector("#logo");
let firstP   = document.querySelector("article p");

// 5. querySelectorAll — returns ALL matches as a NodeList
let allBtns = document.querySelectorAll(".btn");
```

Method	Returns	Best Used For
getElementById(id)	Single element	When a unique id is set.
getElementsByClassName(c)	Live HTMLCollection	All elements with a class.
getElementsByTagName(t)	Live HTMLCollection	All elements of a tag type.
querySelector(sel)	First match	Any CSS selector — modern default.
querySelectorAll(sel)	NodeList	All matches of a CSS selector.

> **Modern advice:** *querySelector* and *querySelectorAll* work everywhere and accept any CSS selector — they are the most flexible choice.

4.2 Looping Through a NodeList

```
let cards = document.querySelectorAll(".card");

cards.forEach(card => {
  card.style.background = "#fff7d6";
});
```

5. Modifying Elements

5.1 Changing Text and HTML

```
<h1 id="title">Hello!</h1>
<p id="msg">Loading…</p>

<script>
  // Plain text
  document.getElementById("title").textContent = "Welcome to BCA";

  // HTML content (allows tags inside)
  document.getElementById("msg").innerHTML =
    "Page loaded <strong>successfully</strong>.";
</script>
```

> ***textContent* vs *innerHTML*** - *textContent* treats the input as plain text — safer. - *innerHTML* parses HTML tags inside the string — more powerful but risky if the input comes from a user (security: XSS attacks).

5.2 Changing Attributes

```
<img id="photo" src="default.jpg" alt="Default">
<a id="link" href="#">Click</a>

<script>
  document.getElementById("photo").src = "campus.jpg";
  document.getElementById("photo").alt = "RVSCET campus";

  document.getElementById("link").href = "https://rvscet.com";
  document.getElementById("link").textContent = "Visit RVSCET";

  // Generic methods that work for any attribute
  let img = document.getElementById("photo");
  img.setAttribute("title", "Hover text");
  console.log(img.getAttribute("alt"));
  img.removeAttribute("title");
</script>
```

5.3 Changing CSS Styles

```
let box = document.getElementById("box");

// Inline style — one property at a time
box.style.color = "white";
box.style.backgroundColor = "navy";      // CSS background-color → backgroundColor
box.style.padding = "20px";
box.style.borderRadius = "8px";

// Better: add or remove a CSS class
box.classList.add("highlight");
box.classList.remove("dimmed");
```

```
box.classList.toggle("active");          // adds if absent, removes if present
console.log(box.classList.contains("active"));   // true/false
```

Naming convention: in JavaScript, CSS hyphens become camelCase: *font-size → fontSize*, *background-color → backgroundColor*.

5.4 Creating and Removing Elements

```
// Create a new element
let newPara = document.createElement("p");
newPara.textContent = "This was added by JavaScript!";
newPara.className = "note";

// Add it to the page (at the end of <body>)
document.body.appendChild(newPara);

// Add at a specific place
let container = document.getElementById("list");
container.prepend(newPara);     // at the start
container.append(newPara);      // at the end

// Remove an element
let oldElement = document.getElementById("warning");
oldElement.remove();

// Replace text inside a container by clearing and adding
container.innerHTML = "";       // empty it
container.appendChild(newPara);
```

5.5 Mini Example — Counter Button

```
<!DOCTYPE html>
<html>
<body>

  <h2>Click counter</h2>
  <p>You have clicked <span id="count">0</span> times.</p>
  <button id="btn">Click me</button>

  <script>
    let count = 0;
    const btn = document.getElementById("btn");
    const countEl = document.getElementById("count");

    btn.onclick = function() {
      count++;
      countEl.textContent = count;
    };
  </script>

</body>
</html>
```

6. Events

An **event** is something that happens on the page that JavaScript can react to: a button click, a key press, the page finishing loading, the mouse moving over an image, and many more.

6.1 Common Events

Event	Triggered when…
click	The user clicks an element.
dblclick	The user double-clicks.
mouseover / *mouseout*	The mouse enters / leaves the element.
keydown / *keyup*	A key is pressed / released.
change	A form input value is changed and committed.
input	A form input value is being typed.
submit	A form is submitted.
focus / *blur*	An input gains / loses focus.
load	The page (or image) has finished loading.
scroll	The page is scrolled.
resize	The browser window is resized.

6.2 Three Ways to Handle an Event

Method 1 — Inline HTML attribute (avoid in real projects):

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

Method 2 — DOM property:

```
<button id="btn">Click</button>
<script>
  document.getElementById("btn").onclick = function() {
    alert("Clicked!");
  };
</script>
```

Method 3 — *addEventListener* (recommended):

```
const btn = document.getElementById("btn");

btn.addEventListener("click", function() {
  alert("Clicked using addEventListener!");
});

// You can attach more than one listener for the same event
btn.addEventListener("click", () => console.log("Also logged"));
```

> **Why prefer *addEventListener*?** - You can attach multiple handlers for the same event. - You can remove a specific handler later with *removeEventListener*. - Keeps HTML and JavaScript cleanly separated.

6.3 The Event Object

When an event fires, the browser passes an **event object** to your handler. It contains useful information about what happened.

```
const btn = document.getElementById("btn");

btn.addEventListener("click", function(event) {
    console.log(event.type);          // "click"
    console.log(event.target);        // the element that was clicked
    console.log(event.clientX, event.clientY);  // mouse coordinates
});

// Keyboard example
document.addEventListener("keydown", function(e) {
    console.log("You pressed:", e.key);
    if (e.key === "Escape") {
        alert("Escape pressed!");
    }
});
```

6.4 Event Example — Background Changer

```
<!DOCTYPE html>
<html>
<body style="text-align: center; padding-top: 100px;">

    <h1 id="msg">Click any button to change the background</h1>
    <button class="colorBtn" data-color="lightblue">Blue</button>
    <button class="colorBtn" data-color="lightgreen">Green</button>
    <button class="colorBtn" data-color="lightyellow">Yellow</button>
    <button class="colorBtn" data-color="lightpink">Pink</button>

    <script>
        const buttons = document.querySelectorAll(".colorBtn");

        buttons.forEach(btn => {
            btn.addEventListener("click", function(e) {
                document.body.style.background = e.target.dataset.color;
                document.getElementById("msg").textContent =
                    "Background changed to " + e.target.textContent;
            });
        });
    </script>

</body>
</html>
```

6.5 *preventDefault()*

Stops the browser's default action for an event. Most often used to prevent a form from submitting before validation finishes.

```
document.querySelector("form").addEventListener("submit", function(e) {
    e.preventDefault();      // stop the form from sending immediately
```

```
  console.log("Form intercepted — running validation…");
});
```

---

7. Form Validation with JavaScript

We saw HTML5's built-in validation in Week 4 (*required*, *pattern*, etc.). For more flexibility, JavaScript can validate forms with custom rules and friendly messages.

7.1 Reading Form Values

```
const nameInput  = document.getElementById("name");
const emailInput = document.getElementById("email");
const ageInput   = document.getElementById("age");

console.log(nameInput.value);
console.log(emailInput.value);
console.log(Number(ageInput.value));   // convert text to number
```

7.2 A Complete Validation Example

```
<!DOCTYPE html>
<html>
<head>
  <style>
    body { font-family: Arial; padding: 20px; max-width: 500px; }
    input { width: 100%; padding: 8px; margin: 4px 0 12px;
            font-size: 16px; }
    button { padding: 10px 18px; font-size: 16px; }
    .error { color: red; font-size: 14px; }
    .success { color: green; font-weight: bold; }
  </style>
</head>
<body>

  <h2>Registration Form</h2>

  <form id="regForm">
    <label>Full Name:</label>
    <input type="text" id="name">
    <span class="error" id="nameErr"></span>

    <label>Email:</label>
    <input type="text" id="email">
    <span class="error" id="emailErr"></span>

    <label>Age:</label>
    <input type="number" id="age">
    <span class="error" id="ageErr"></span>

    <label>Password:</label>
```

```
    <input type="password" id="pwd">
    <span class="error" id="pwdErr"></span>

    <button type="submit">Register</button>
</form>

<p id="result" class="success"></p>

<script>
    const form = document.getElementById("regForm");

    form.addEventListener("submit", function(e) {
        e.preventDefault();        // stop default submission

        // Clear old errors
        document.querySelectorAll(".error").forEach(el => el.textContent = "");
        document.getElementById("result").textContent = "";

        const name  = document.getElementById("name").value.trim();
        const email = document.getElementById("email").value.trim();
        const age   = Number(document.getElementById("age").value);
        const pwd   = document.getElementById("pwd").value;

        let isValid = true;

        // Name — at least 3 characters
        if (name.length < 3) {
            document.getElementById("nameErr").textContent =
                "Name must be at least 3 characters.";
            isValid = false;
        }

        // Email — must contain @ and .
        if (!email.includes("@") || !email.includes(".")) {
            document.getElementById("emailErr").textContent =
                "Please enter a valid email.";
            isValid = false;
        }

        // Age — between 16 and 60
        if (isNaN(age) || age < 16 || age > 60) {
            document.getElementById("ageErr").textContent =
                "Age must be between 16 and 60.";
            isValid = false;
        }

        // Password — at least 8 characters with one digit
        if (pwd.length < 8 || !/\d/.test(pwd)) {
            document.getElementById("pwdErr").textContent =
                "Password must be 8+ characters and contain a digit.";
```

```
            isValid = false;
        }

        if (isValid) {
            document.getElementById("result").textContent =
                `Welcome, ${name}! Registration successful.`;
            form.reset();
        }
    });
</script>

</body>
</html>
```

Always validate twice: once on the client (this code, for friendly UX) and once on the server (Week 10, for security). Client-side validation can be bypassed.

---

8. JSON

8.1 What is JSON?

JSON stands for **JavaScript Object Notation**. It is a lightweight text format used to send and store structured data — for example, when a web page asks the server "give me the latest news" and the server replies with a JSON document. JSON looks almost exactly like a JavaScript object, but every key must be wrapped in **double quotes** and only certain value types are allowed.

```
{
  "name": "Aman Kumar",
  "roll": 101,
  "isHosteler": true,
  "subjects": ["Web Tech", "DSA", "Maths"],
  "address": {
    "city": "Jamshedpur",
    "state": "Jharkhand"
  }
}
```

8.2 JSON vs JavaScript Object

Aspect	JavaScript Object	JSON
Keys	May or may not have quotes	**Must** be in double quotes
Strings	Single or double quotes	**Double quotes only**
Methods (functions)	Allowed	**Not allowed**
Comments	Allowed	**Not allowed**
Trailing commas	Allowed	**Not allowed**
File extension	*.js*	*.json*
Used for	Programs running in memory	Data exchange / storage

8.3 Allowed JSON Value Types

- string → *"Aman"*
- number → *42*, *3.14*
- boolean → *true*, *false*
- null → *null*
- array → *["a", "b", "c"]*
- object → *{ "key": "value" }*

8.4 Working with JSON in JavaScript

JavaScript provides two built-in methods on the global *JSON* object:

```
const student = {
  name: "Aman",
  roll: 101,
  course: "BCA"
};

// 1. Convert object → JSON string
const jsonText = JSON.stringify(student);
console.log(jsonText);
// {"name":"Aman","roll":101,"course":"BCA"}

// Pretty-print with 2 spaces of indentation
console.log(JSON.stringify(student, null, 2));

// 2. Convert JSON string → object
const text = '{"name":"Riya","roll":102,"course":"BCA"}';
const obj  = JSON.parse(text);

console.log(obj.name);    // "Riya"
console.log(obj.roll);    // 102
```

Method	What it does
JSON.stringify(obj)	Object → JSON string.
JSON.parse(text)	JSON string → object.

8.5 A Realistic Example

Imagine a server sending a list of students as JSON. Here's how we'd parse and display it:

```
const responseText = `[
  { "name": "Aman",  "marks": 78 },
  { "name": "Riya",  "marks": 92 },
  { "name": "Vivek", "marks": 67 }
]`;

const students = JSON.parse(responseText);

students.forEach(s => {
  console.log(`${s.name}: ${s.marks}`);
});
```

```
// Output:
// Aman: 78
// Riya: 92
// Vivek: 67
```

> **Where you'll meet JSON next:** - Week 10 (PHP) — server reads form data and replies in JSON. - Week 11+ (Servlets / web services) — APIs send and receive JSON. - Anywhere a webpage talks to a backend — JSON is the standard.

---

9. Worked Example — Mini To-Do List

This example combines everything from this week: arrays, objects, the DOM, events, and JSON (for saving). It's a realistic mini-app.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>BCA To-Do</title>
  <style>
    body { font-family: Arial; padding: 20px; max-width: 500px; }
    input, button { padding: 8px; font-size: 16px; }
    ul { list-style: none; padding: 0; }
    li { background: #f0f8ff; padding: 10px; margin: 5px 0;
      border-radius: 6px; display: flex;
      justify-content: space-between; align-items: center; }
    li.done { text-decoration: line-through; opacity: 0.6; }
    button.del { background: #dc2626; color: white; border: none;
          padding: 4px 8px; cursor: pointer; }
  </style>
</head>
<body>

  <h1>My To-Do List</h1>
  <input type="text" id="taskInput" placeholder="What needs to be done?">
  <button id="addBtn">Add Task</button>

  <ul id="taskList"></ul>

  <p>Total tasks: <span id="count">0</span></p>

  <script>
    // Array of objects — each task has text and a done flag
    let tasks = [];

    const input = document.getElementById("taskInput");
    const addBtn = document.getElementById("addBtn");
    const list = document.getElementById("taskList");
    const count = document.getElementById("count");
```

```
    function render() {
      list.innerHTML = "";
      tasks.forEach((task, index) => {
        const li = document.createElement("li");
        if (task.done) li.classList.add("done");

        li.innerHTML = `
          <span>${task.text}</span>
          <button class="del">Delete</button>
        `;

        // Toggle done on click of text
        li.querySelector("span").addEventListener("click", () => {
          tasks[index].done = !tasks[index].done;
          render();
        });

        // Delete button
        li.querySelector(".del").addEventListener("click", () => {
          tasks.splice(index, 1);
          render();
        });

        list.appendChild(li);
      });
      count.textContent = tasks.length;

      // (Bonus) Save tasks as JSON — could be sent to a server
      console.log("Saved JSON:", JSON.stringify(tasks));
    }

    addBtn.addEventListener("click", () => {
      const text = input.value.trim();
      if (text === "") return;
      tasks.push({ text: text, done: false });
      input.value = "";
      render();
    });

    // Press Enter to add
    input.addEventListener("keydown", e => {
      if (e.key === "Enter") addBtn.click();
    });

    render();
  </script>

</body>
</html>
```

Try this in your lab 1. Save as *todo.html* and open in Chrome. 2. Type a task → click **Add Task** (or press Enter). 3. Click on the task text → it's marked as done (strikethrough). 4. Click **Delete** → it's removed. 5. Open DevTools (F12) → Console — you'll see the task list saved as JSON.

---

10. Summary

- **Arrays** are ordered lists indexed from 0; use methods like *push*, *pop*, *forEach*, *map*, *filter*, *reduce*.
- **Objects** are key-value pairs accessed via dot or bracket notation; functions inside an object are called methods.
- The **DOM** is the in-memory tree of an HTML page; JavaScript reads and modifies it.
- Select elements with *getElementById*, *getElementsByClassName*, *querySelector*, *querySelectorAll*.
- Modify content (*textContent*, *innerHTML*), attributes (*setAttribute*), and styles (*style*, *classList*).
- Create and remove elements with *createElement*, *appendChild*, and *remove*.
- Handle user actions with **events**; prefer *addEventListener* for flexibility.
- The event object provides info like *event.target*, *event.key*, *event.preventDefault()*.
- **Validate forms** in JavaScript by reading *.value*, checking conditions, and showing error messages.
- **JSON** is a text format for structured data; convert with *JSON.stringify()* and *JSON.parse()*.

---

11. Practice Questions

A. Short Answer (2 marks each)

6. Write JavaScript to create an array of three subject names and print the second one.
7. What is the difference between *textContent* and *innerHTML*?
8. Write the syntax of *addEventListener* with one example.
9. List any three array methods with one-line code examples.
10. Differentiate between a JavaScript object and a JSON document.
11. What does *event.preventDefault()* do?
12. Write the JavaScript to convert this object to a JSON string: *{ name: "Aman", marks: 90 }*.

B. Long Answer (5–10 marks each)

13. Explain at least eight commonly used array methods with code examples for each.
14. Explain the DOM with a tree diagram. Show how to select an element by id, by class, and using a CSS selector.
15. Describe at least five common DOM events with one HTML+JS example for each.
16. Discuss form validation in JavaScript using *addEventListener("submit")* and *preventDefault()*. Write a full example for a login form (email + password) with error messages.
17. Explain JSON, its rules, and how it differs from a JavaScript object. Show how *JSON.stringify* and *JSON.parse* are used with a worked example.

C. Lab / Hands-On Tasks

18. Build a page with a button labelled "Show Time". When clicked, it displays the current date and time inside a <*p*> element using *new Date()*.
19. Build an image slider — three images with **Previous** and **Next** buttons that change the displayed image. Use an array of image filenames and an index variable.

20. Build a colour picker — three sliders (red, green, blue, each 0–255) that update a <*div*>'s background colour in real time using the *input* event.
21. Build a contact list app: enter name and phone, click Add, the contact appears in a list. Each contact has a Delete button. Save the list to a JSON string and print it in the console after every change.
22. Take the registration form from Week 4 and add full JavaScript validation: at least 4 fields, each with a custom error message that appears next to the field, no use of HTML5 *required*.

WEEK 10 — PHP / SERVER-SIDE BASICS

Course Outcome: CO4 | Topics: PHP intro, syntax, variables, control flow, functions, arrays, forms, sessions

Learning Objectives

By the end of this week, students will be able to:

- Explain what server-side programming is and how PHP fits into the web stack.
- Set up XAMPP locally and run their first PHP page.
- Write PHP code with variables, data types, operators, and control flow.
- Use PHP arrays (indexed, associative, multidimensional) and define functions.
- Receive form data on the server using $_GET and $_POST.
- Track a logged-in user across pages using sessions and cookies.
- Build a complete login/registration flow that talks to a server-side script.

1. Introduction to Server-Side Programming

1.1 Client-Side vs Server-Side

Until Week 9, all the code we wrote (HTML, CSS, JavaScript) ran inside the user's browser — this is called CLIENT-SIDE code. Anyone can right-click and view the source. Client-side code can build interactive pages but cannot save data permanently, send email, or check passwords against a database.

Real web applications also need a SERVER — a computer somewhere on the internet that stores users, processes payments, and replies to requests. Code that runs on the server is called SERVER-SIDE code. PHP is one of the most popular server-side languages.

Aspect	Client-side (Browser)	Server-side (Server)
Languages	HTML, CSS, JavaScript	PHP, Java/JSP, Python, Node.js, ...
Runs on	User's browser	The web server (Apache, Nginx, ...)
Visible to user?	Yes — anyone can view source	No — only the output is sent to the browser
Can access database?	No (directly)	Yes

Aspect	Client-side (Browser)	Server-side (Server)
Best for	Interactivity, animations, validation	Storing data, login, payments, business logic

1.2 What is PHP?

PHP stands for HYPERTEXT PREPROCESSOR (the name is a recursive acronym). It is a server-side scripting language created by Rasmus Lerdorf in 1994. PHP code is embedded inside HTML files — the server runs the PHP first, and sends only the resulting HTML to the browser.

1.3 Why PHP?

- Free and open-source — runs on Windows, Linux, and Mac.
- Designed specifically for the web — easy to mix with HTML.
- Powers a huge share of the internet — WordPress, Wikipedia, Facebook (originally), and millions of small sites.
- Excellent MySQL support — perfect for student database projects.
- Beginner-friendly — minimal setup, plenty of tutorials, large community.

1.4 How a PHP Page is Served

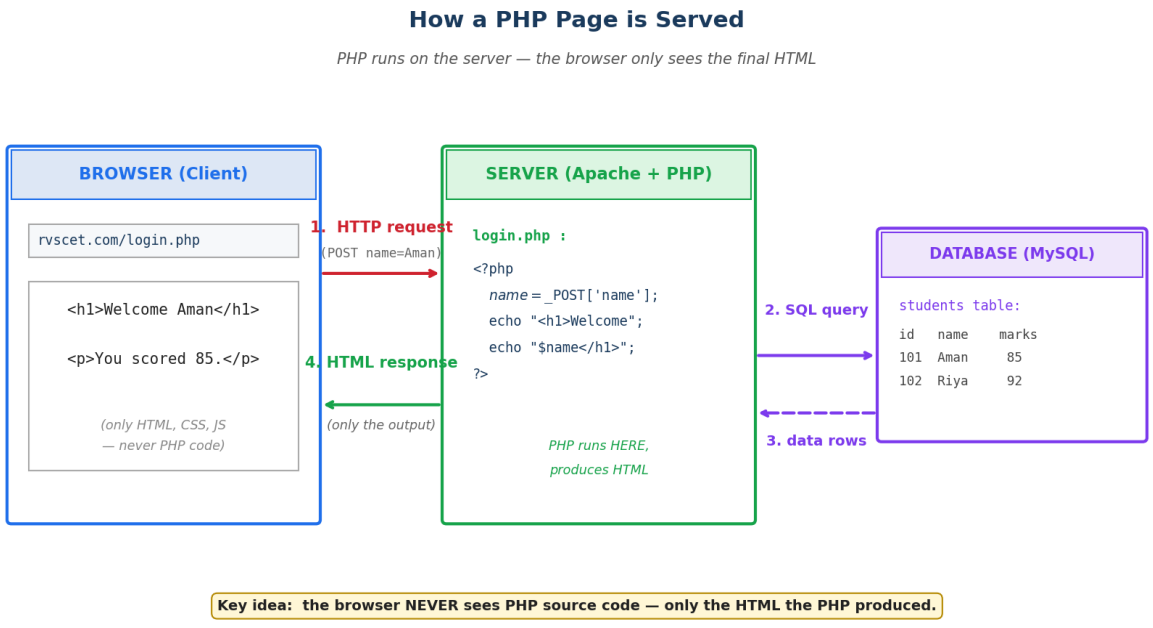


Figure 10.1 — The browser sends a request, the server runs the PHP and queries the database if needed, and only the resulting HTML comes back.

Real-world use cases of PHP

- User registration and login systems
- E-commerce websites — shopping carts, orders, payment integration
- Content Management Systems (WordPress, Joomla, Drupal)
- Online exam portals and result publishing
- Hostel and library management systems for colleges like RVSCET
- REST APIs for mobile and SPA front-ends

2. Setting Up the PHP Environment

2.1 XAMPP — All-in-One PHP Stack

To run PHP on your computer, you need three things working together: a web server (Apache), the PHP language interpreter, and a database (MySQL/MariaDB). XAMPP is a free package that installs all three in one go. The name stands for: X (cross-platform), Apache, MariaDB, PHP, Perl.

2.2 Installation Steps

- Go to apachefriends.org and download XAMPP for Windows (or your OS).
- Run the installer with default options. Install in C:\xampp.
- Open the XAMPP Control Panel.
- Click START next to Apache and MySQL — both should turn green.
- Open Chrome and visit http://localhost/ — you should see the XAMPP welcome page.

2.3 Where to Save Your PHP Files

All PHP files must be inside the htdocs folder to be served. The default path is C:\xampp\htdocs. Files inside this folder are accessible at http://localhost/.

Folder structure for PHP projects

```
C:\xampp\htdocs\            <-- web root
    |
    |-- index.html          accessible at http://localhost/
    |-- bca\
    |   |-- hello.php       http://localhost/bca/hello.php
    |   |-- login.php       http://localhost/bca/login.php
    |   |-- styles.css
```

Important

Do NOT open PHP files by double-clicking them — that opens them in Notepad or shows raw code in the browser.

Always start Apache in XAMPP first, then visit http://localhost/yourfile.php in the browser. The server runs the PHP and shows you the output.

3. PHP Syntax Basics

3.1 PHP Tags

PHP code is wrapped in special tags. Anything outside the tags is treated as plain HTML.

Example 3.1 — hello.php (save in htdocs and visit http://localhost/hello.php)

```
<!DOCTYPE html>
<html>
<body>
  <h1>My First PHP Page</h1>

  <?php
    echo "<p>Hello from PHP!</p>";
  ?>

  <p>This is plain HTML again.</p>
</body>
</html>
```

> Output in browser:

My First PHP Page
Hello from PHP!
This is plain HTML again.

3.2 echo and print

Both echo and print are used to output text. echo is slightly faster and is preferred. Statements end with a semicolon.

Example 3.2 — echo and print

```
<?php
  echo "Hello, World!";
  echo "<br>";
  echo "Welcome to BCA ", "3rd Semester";   // multiple values

  print("Print also works.");
?>
```

3.3 Comments

Example 3.3 — Three styles of PHP comments

```php
<?php
   // This is a single-line comment
   # This is also a single-line comment (rarely used)

   /* This is a
     multi-line comment */

   echo "Comments don't appear in output.";
?>
```

3.4 Mixing PHP with HTML

PHP shines when mixed with HTML. You can drop in and out of PHP as many times as needed.

Example 3.4 — Mixing PHP with HTML using shorter <?php ?> blocks

```php
<?php $name = "Aman"; $marks = 85; ?>

<h1>Result for <?php echo $name; ?></h1>
<p>You scored <?php echo $marks; ?> out of 100.</p>

<?php if ($marks >= 40) { ?>
   <p style='color:green'>Status: PASS</p>
<?php } else { ?>
   <p style='color:red'>Status: FAIL</p>
<?php } ?>
```

4. Variables and Data Types

4.1 Declaring Variables

PHP variables start with a $ sign followed by a name. PHP is dynamically typed — you do NOT declare the type. The same variable can hold different types over time.

Example 4.1 — Declaring variables of different types

```php
<?php
   $name = "Aman Kumar";      // string
   $age = 21;                 // integer
   $marks = 78.5;             // float
   $isHosteler = true;        // boolean
   $courses = ["BCA", "MCA"];  // array
   $address = null;           // null

   echo $name;      // Aman Kumar
   echo $age;       // 21
?>
```

Type	Description

String	Text in single or double quotes — "Aman", 'BCA'.
Integer	Whole numbers — 42, -17, 0.
Float	Numbers with decimals — 3.14, 78.5.
Boolean	true or false.
Array	Ordered or named collection of values.
NULL	A variable that has been assigned no value.
Object	Instance of a class (covered in advanced courses).

4.2 Naming Rules

- Must start with $ followed by a letter or underscore.
- Cannot start with a number — $1name is invalid.
- Case-sensitive — $Name and $name are different variables.
- Convention: use snake_case ($first_name) or camelCase ($firstName).

4.3 Strings — Single vs Double Quotes

Example 4.2 — String basics

```
<?php
   $name = "Riya";

   // Double quotes — variables ARE replaced
   echo "Hello, $name!";         // Hello, Riya!

   // Single quotes — variables NOT replaced
   echo 'Hello, $name!';         // Hello, $name!

   // String concatenation with the dot operator
   echo "Hello, " . $name . "!";  // Hello, Riya!

   // Useful string functions
   echo strlen($name);           // 4 (length)
   echo strtoupper($name);       // RIYA
   echo strtolower($name);       // riya
   echo str_replace("Riya", "Aman", "Hi Riya");  // Hi Aman
?>
```

5. Operators

5.1 Arithmetic

Example 5.1 — Arithmetic operators

```
<?php
   $a = 10; $b = 3;
   echo $a + $b;     // 13
   echo $a - $b;     // 7
   echo $a * $b;     // 30
   echo $a / $b;     // 3.333...
   echo $a % $b;     // 1   (modulus)
   echo $a ** $b;    // 1000  (a to the power b)
?>
```

5.2 String Concatenation

PHP uses the dot (.) operator to join strings, NOT the plus sign. This is a common difference from JavaScript.

Example 5.2 — String concatenation

```
<?php
   $first = "Aman";
   $last = "Kumar";
   $full = $first . " " . $last;
   echo $full;         // Aman Kumar

   // With assignment
   $msg = "Hello";
   $msg .= ", World!";   // append
   echo $msg;          // Hello, World!
?>
```

5.3 Comparison and Logical

Example 5.3 — Comparison and logical operators

```
<?php
   echo var_export(5 == "5");   // true   (loose — converts type)
   echo var_export(5 === "5");  // false  (strict — types must match)
   echo var_export(5 != 3);      // true
   echo var_export(5 !== "5");  // true
   echo var_export(5 > 3);       // true
   echo var_export(5 <= 5);      // true

   $age = 20; $hasID = true;
   echo var_export($age >= 18 && $hasID);   // true   AND
   echo var_export($age >= 18 || $hasID);   // true   OR
   echo var_export(!$hasID);                // false  NOT
?>
```

Tip — always prefer === and !==

Loose comparison (==) does automatic type conversion that can produce surprising results — for example, 0 == 'hello' is true in older versions of PHP. Strict comparison (===) checks both value AND type, which is safer.

6. Control Flow

6.1 if / else if / else

Example 6.1 — if / elseif / else

```
<?php
   $marks = 78;

   if ($marks >= 90) {
      echo "Grade A+";
   } elseif ($marks >= 80) {
      echo "Grade A";
   } elseif ($marks >= 70) {
      echo "Grade B";
   } elseif ($marks >= 40) {
      echo "Grade C";
   } else {
      echo "Fail";
   }
?>
```

6.2 switch

Example 6.2 — switch

```
<?php
   $day = 3;

   switch ($day) {
      case 1: echo "Monday";    break;
      case 2: echo "Tuesday";   break;
      case 3: echo "Wednesday"; break;
      case 4: echo "Thursday";  break;
      case 5: echo "Friday";    break;
      default: echo "Weekend";
   }
?>
```

6.3 Loops

Example 6.3 — All four PHP loops

```php
<?php
    // for loop
    for ($i = 1; $i <= 5; $i++) {
        echo "Number $i <br>";
    }

    // while loop
    $n = 1;
    while ($n <= 3) {
        echo "n = $n <br>";
        $n++;
    }

    // do...while loop (runs at least once)
    $x = 10;
    do {
        echo "x = $x <br>";
        $x++;
    } while ($x <= 5);

    // foreach — best for arrays
    $subjects = ["Web Tech", "DSA", "Maths"];
    foreach ($subjects as $s) {
        echo "$s <br>";
    }
?>
```

7. Arrays in PHP

7.1 Indexed Arrays

Like JavaScript arrays — items accessed by their position, starting from 0.

Example 7.1 — Indexed arrays

```php
<?php
    $subjects = ["Web Tech", "DSA", "Maths", "English"];
    // older syntax: array("Web Tech", "DSA", ...)

    echo $subjects[0];          // Web Tech
    echo $subjects[2];          // Maths
    echo count($subjects);      // 4

    // Add to the end
    $subjects[] = "PHP";
    array_push($subjects, "MySQL");

    print_r($subjects);
```

```
?>
```

7.2 Associative Arrays (Key-Value Pairs)

Like JavaScript objects — items accessed by a string key.

Example 7.2 — Associative arrays

```
<?php
    $student = [
        "name"   => "Aman Kumar",
        "roll"   => 101,
        "course" => "BCA",
        "marks"  => 85
    ];

    echo $student["name"];      // Aman Kumar
    echo $student["marks"];     // 85

    // Add or change
    $student["semester"] = 3;
    $student["marks"] = 90;
?>
```

7.3 Multidimensional Arrays

Example 7.3 — Array of student records

```
<?php
    // Array of student records
    $students = [
        ["name" => "Aman",  "marks" => 85],
        ["name" => "Riya",  "marks" => 92],
        ["name" => "Vivek", "marks" => 67]
    ];

    // Access individual cells
    echo $students[1]["name"];   // Riya

    // Loop through all records
    foreach ($students as $s) {
        echo "$s[name] scored $s[marks] <br>";
    }
?>
```

7.4 Useful Array Functions

Function	What it does
count($arr)	Returns the number of elements.
array_push($arr, $x)	Add to the end.
array_pop($arr)	Remove and return the last element.
array_unshift($arr, $x)	Add to the beginning.
array_shift($arr)	Remove and return the first element.
sort($arr)	Sort in ascending order.
rsort($arr)	Sort in descending order.
in_array($x, $arr)	Check if a value is present (returns true/false).
array_keys($arr)	Get all keys of an associative array.
array_values($arr)	Get all values of an associative array.
implode(",", $arr)	Join array into a string.
explode(",", $str)	Split a string into an array.

8. Functions

8.1 Defining and Calling

Example 8.1 — A simple function

```
<?php
   // Define
   function greet($name) {
      return "Hello, $name!";
   }

   // Call
   echo greet("Aman");        // Hello, Aman!
   echo greet("Riya");        // Hello, Riya!
?>
```

8.2 Default Parameters

Example 8.2 — Default parameter value

```
<?php
   function welcome($name = "Guest") {
      return "Welcome, $name!";
```

```
    }

    echo welcome("Aman");       // Welcome, Aman!
    echo welcome();             // Welcome, Guest!
?>
```

8.3 Variable Scope

Variables declared inside a function are LOCAL to that function. Variables outside are GLOBAL. To use a global variable inside a function, declare it with the global keyword.

Example 8.3 — Variable scope

```
<?php
    $college = "RVSCET";        // global

    function printCollege() {
        global $college;        // import the global
        echo $college;
    }

    printCollege();             // RVSCET
?>
```

8.4 A Practical Function — Calculate Grade

Example 8.4 — Reusable grade calculation function

```
<?php
    function calculateGrade($marks) {
        if ($marks >= 90)      return "A+";
        elseif ($marks >= 80)  return "A";
        elseif ($marks >= 70)  return "B";
        elseif ($marks >= 60)  return "C";
        elseif ($marks >= 40)  return "D";
        else                   return "F";
    }

    echo calculateGrade(95);    // A+
    echo calculateGrade(75);    // B
    echo calculateGrade(35);    // F
?>
```

9. Receiving Form Data

9.1 GET vs POST — Recap and Diagram

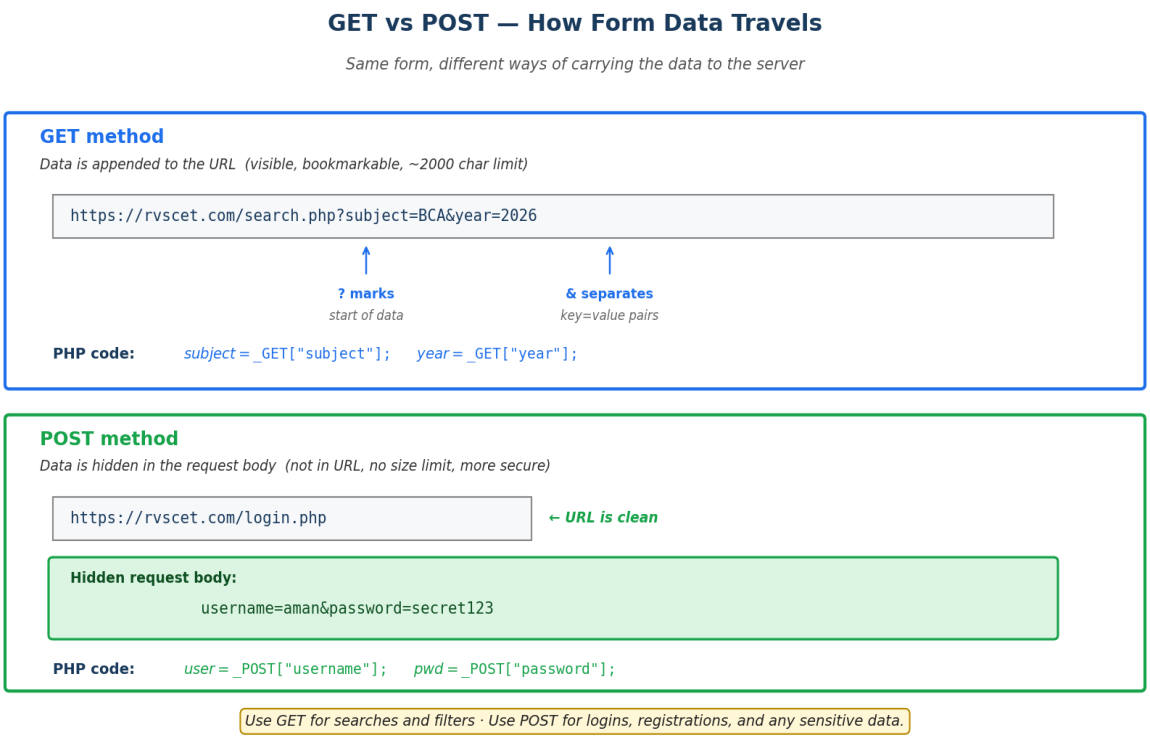


Figure 10.2 — GET puts form data in the URL; POST hides it in the request body.

9.2 Receiving GET Data

Example 9.1a — search-form.html

```
<!-- File: search-form.html -->
<form action="search.php" method="GET">
  <input type="text" name="keyword">
  <input type="text" name="category">
  <button type="submit">Search</button>
</form>
```

Example 9.1b — search.php receives the GET data

```
<?php
  // File: search.php
  // URL becomes: search.php?keyword=html&category=tutorial

  $keyword  = $_GET["keyword"];
  $category = $_GET["category"];

  echo "<h2>Search Results</h2>";
  echo "You searched for '<b>$keyword</b>' in '<b>$category</b>'.";
?>
```

9.3 Receiving POST Data

Example 9.2a — login-form.html

```
<!-- File: login-form.html -->
<form action="login.php" method="POST">
  <label>Username:</label>
  <input type="text" name="username" required>

  <label>Password:</label>
  <input type="password" name="password" required>

  <button type="submit">Login</button>
</form>
```

Example 9.2b — login.php receives the POST data

```
<?php
  // File: login.php
  $user = $_POST["username"];
  $pwd  = $_POST["password"];

  if ($user === "admin" && $pwd === "bca123") {
    echo "<h2>Welcome, $user!</h2>";
  } else {
    echo "<p style='color:red'>Invalid username or password.</p>";
  }
?>
```

9.4 $_REQUEST and Checking the Method

Example 9.3 — $_REQUEST and detecting the method

```
<?php
  // $_REQUEST contains both GET and POST values
  $name = $_REQUEST["name"];

  // Check which method was used
  if ($_SERVER["REQUEST_METHOD"] === "POST") {
    echo "Form was submitted via POST";
  }
?>
```

9.5 Sanitising Input — Always!

Never trust user input. Always clean it before using it. Two essential PHP functions:

Example 9.4 — Sanitising user input

```
<?php
  // 1. Trim removes leading/trailing whitespace
  $name = trim($_POST["name"]);
```

```
    // 2. htmlspecialchars converts < > & ' " to safe HTML entities
    //    — prevents XSS attacks where users inject <script> tags
    $safe = htmlspecialchars($name);

    echo "Hello, $safe!";
?>
```

10. Sessions and Cookies

10.1 The Problem — HTTP is Stateless

Each HTTP request is independent. The server has no memory of past requests by the same user. So once a user logs in, how does the server remember them on the next page? The answer is sessions and cookies.

10.2 Sessions

A session stores data on the server and gives the browser a unique session ID (a cookie). Every time the browser sends a request, the server uses the ID to find the matching data. Sessions are the standard way to track logged-in users.

Example 10.1a — login_check.php sets session values

```
<?php
    // Page 1: login_check.php — set the session
    session_start();   // MUST be called before any output

    // After verifying the user...
    $_SESSION["username"] = "Aman";
    $_SESSION["role"] = "student";

    header("Location: dashboard.php");
    exit();
?>
```

Example 10.1b — dashboard.php reads the session

```
<?php
    // Page 2: dashboard.php — read the session
    session_start();

    // Protect the page — redirect if not logged in
    if (!isset($_SESSION["username"])) {
        header("Location: login-form.html");
        exit();
    }

    $name = $_SESSION["username"];
```

```
    echo "<h2>Welcome back, $name!</h2>";
    echo "<a href='logout.php'>Logout</a>";
?>
```

Example 10.1c — logout.php destroys the session

```
<?php
    // Page 3: logout.php — destroy the session
    session_start();
    session_unset();     // remove all session variables
    session_destroy();   // close the session
    header("Location: login-form.html");
?>
```

10.3 Cookies

A cookie stores data in the user's browser, not on the server. Cookies are sent with every request to the same domain. They are useful for remembering preferences (theme, language, last visit) — NOT for sensitive data.

Example 10.2 — Setting, reading, and deleting cookies

```
<?php
    // Set a cookie that lasts 7 days
    setcookie("username", "Aman", time() + (7 * 24 * 60 * 60), "/");

    // Read a cookie (note: only on the NEXT request after setting)
    if (isset($_COOKIE["username"])) {
        echo "Welcome back, " . $_COOKIE["username"];
    } else {
        echo "First visit!";
    }

    // Delete by setting expiry in the past
    setcookie("username", "", time() - 3600, "/");
?>
```

10.4 Sessions vs Cookies

Aspect	Session ($_SESSION)	Cookie ($_COOKIE)
Where stored	On the server.	In the user's browser.
Visible to user?	No.	Yes — can be inspected and edited.
Size limit	Large (server memory).	About 4 KB per cookie.
Default lifetime	Until browser is closed.	Until expiry date or manual delete.
Best for	Logged-in user info, cart data.	Remembering preferences, last visit.

11. Putting It All Together — The PHP Lifecycle

Now that we have seen the pieces, here is the complete journey from a button click to a rendered page.

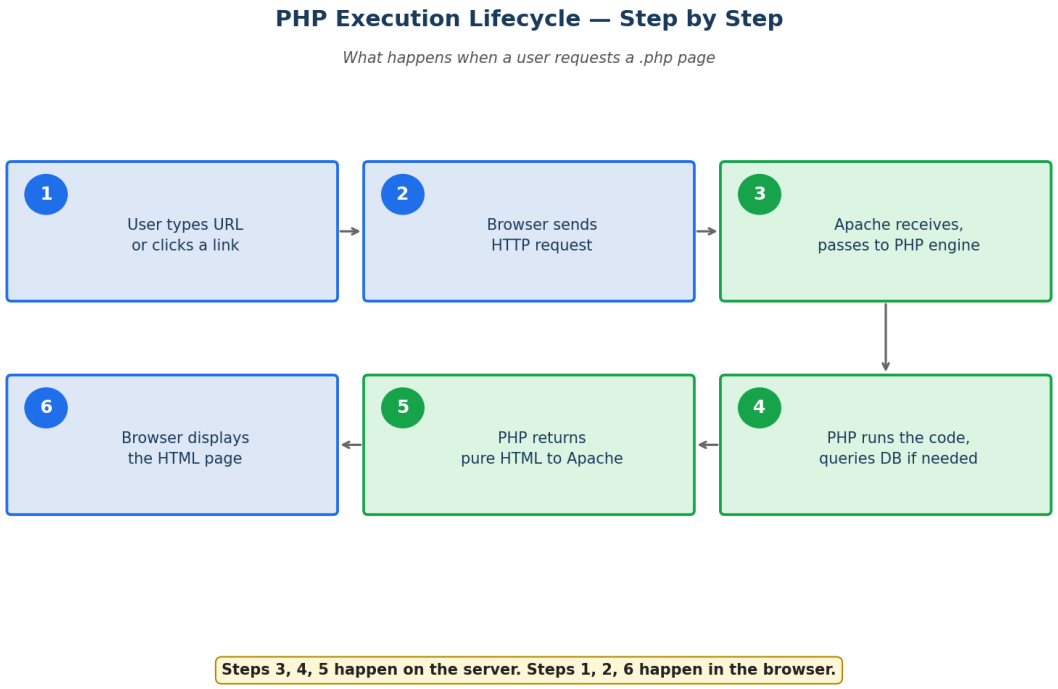


Figure 10.3 — The six-step lifecycle of every PHP request.

12. Worked Example — A Complete Login Flow

This example combines forms, POST data, sessions, and protected pages. Save all four files in the same htdocs subfolder.

Example 12.1a — login.html

```
<!-- File 1: login.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
  <style>
    body { font-family: Arial; padding: 30px; max-width: 400px; }
    input { width: 100%; padding: 8px; margin: 6px 0 14px; }
    button { padding: 10px 20px; background: #1f6feb;
          color: white; border: none; cursor: pointer; }
  </style>
</head>
<body>
  <h2>BCA Portal — Login</h2>
  <form action="check.php" method="POST">
```

```
    <label>Username:</label>
    <input type="text" name="username" required>

    <label>Password:</label>
    <input type="password" name="password" required>

    <button type="submit">Login</button>
  </form>
</body>
</html>
```

Example 12.1b — check.php

```
<?php
  // File 2: check.php
  session_start();

  $user = trim($_POST["username"]);
  $pwd  = $_POST["password"];

  // In a real app, look up in a database (Week 13)
  $validUsers = [
    "aman"  => "bca123",
    "riya"  => "riya456",
    "admin" => "admin789"
  ];

  if (isset($validUsers[$user]) && $validUsers[$user] === $pwd) {
    $_SESSION["username"] = $user;
    $_SESSION["loginTime"] = date("Y-m-d H:i:s");
    header("Location: dashboard.php");
    exit();
  } else {
    echo "<h2>Invalid credentials.</h2>";
    echo "<a href='login.html'>Try again</a>";
  }
?>
```

Example 12.1c — dashboard.php

```
<?php
  // File 3: dashboard.php
  session_start();

  if (!isset($_SESSION["username"])) {
    header("Location: login.html");
    exit();
  }

  $name = htmlspecialchars($_SESSION["username"]);
```

```
    $time = $_SESSION["loginTime"];
?>
<!DOCTYPE html>
<html>
<head>
    <title>Dashboard</title>
    <style>
        body { font-family: Arial; padding: 30px; }
        .card { background: #f4f6f8; padding: 20px; border-radius: 8px; }
    </style>
</head>
<body>
    <h1>Welcome, <?php echo $name; ?>!</h1>
    <div class="card">
        <p>You logged in at <?php echo $time; ?>.</p>
        <p>This page is protected — only logged-in users can view it.</p>
        <a href="logout.php">Logout</a>
    </div>
</body>
</html>
```

Example 12.1d — logout.php

```
<?php
    // File 4: logout.php
    session_start();
    session_unset();
    session_destroy();
    header("Location: login.html");
    exit();
?>
```

Try this in your lab

1. Save all four files in C:\xampp\htdocs\login\.
2. Start Apache from XAMPP Control Panel.
3. Visit http://localhost/login/login.html in Chrome.
4. Try logging in with username 'aman' and password 'bca123' — it should redirect to the dashboard.
5. Try visiting http://localhost/login/dashboard.php directly without logging in — it should redirect you back to the login form.
6. After logging in, click Logout — you should be sent back to the login page and your session destroyed.

13. Summary of Week 10

- PHP is a server-side language — runs on the server before sending HTML to the browser.
- Set up a local environment using XAMPP; save files in C:\xampp\htdocs\.

- PHP code lives between <?php and ?> tags; statements end with semicolons.
- Variables start with $; PHP is dynamically typed — String, Integer, Float, Boolean, Array, NULL.
- String concatenation uses the dot operator (.), not the plus sign.
- Use === for safer (strict) comparisons.
- Control flow: if / elseif / else, switch, for, while, do-while, foreach.
- Three array types: indexed, associative, multidimensional. Plenty of built-in array functions.
- Receive form data with $_GET (URL) and $_POST (request body).
- Always sanitise user input with trim() and htmlspecialchars().
- Sessions ($_SESSION) keep server-side data per user; cookies ($_COOKIE) store small data in the browser.
- Use sessions for login state; redirect unauthorised users with header("Location: ...").

14. Practice Questions

A. Short Answer (2 marks each)

1. Differentiate between client-side and server-side scripting with one example of each.

2. What is the role of XAMPP and which folder must PHP files be saved in?

3. Write the PHP code to declare a variable holding your name and print 'Hello, NAME!'.

4. Explain the difference between the dot (.) operator and the plus (+) sign in PHP strings.

5. Differentiate between $_GET, $_POST, and $_REQUEST.

6. What is the purpose of session_start() and where should it appear?

7. Differentiate between sessions and cookies in two points.

B. Long Answer (5–10 marks each)

1. Explain how a PHP page is processed by the server with a neat diagram. Show the role of Apache, PHP, and (optionally) MySQL.

2. Describe arrays in PHP. Differentiate between indexed, associative, and multidimensional arrays with code examples.

3. Explain the difference between GET and POST methods. Write a complete HTML form and the matching PHP file that uses POST to receive and display the data.

4. What are PHP sessions? Write a complete login system using sessions: login form, session-set page, protected dashboard, and logout page.

5. Describe at least eight commonly used PHP string and array functions with code examples.

C. Lab / Hands-On Tasks

1. Set up XAMPP and create a hello.php page that prints today's date and a greeting based on the time of day (Good Morning / Afternoon / Evening). Use PHP's date() function.

2. Build a tiny calculator: an HTML form with two number fields and a dropdown of +, -, *, /. The PHP file receives the values via POST and shows the result.

3. Create a marks-grading system in PHP: a form takes the student's name and marks, the PHP page calculates the grade using a function and prints a personalised result.

4. Build a registration form with five fields (name, email, mobile, age, course). On submit, PHP should validate (all required, age between 16 and 60, email contains @, mobile is 10 digits) and show errors or a success page.

5. Extend the worked example login system: replace the hard-coded user list with an array of associative arrays (name, password, role) and show different dashboards for 'student' vs 'admin' roles.

WEEK 11 — JAVA SERVLETS

Course Outcome: CO4 | Topics: Servlet intro, lifecycle, doGet/doPost, request, response, sessions, deployment

Learning Objectives

By the end of this week, students will be able to:

- Define a servlet and explain why it is needed for server-side Java web programming.
- Describe the architecture of a servlet-based application — browser, Tomcat, servlet, database.
- List the three main lifecycle methods of a servlet — init(), service(), destroy() — and explain when each runs.
- Write a servlet using doGet() and doPost() to handle HTTP GET and POST requests.
- Read form data from HttpServletRequest and write HTML to HttpServletResponse.
- Track logged-in users using HttpSession.
- Configure servlet URL mappings using both web.xml and the @WebServlet annotation.
- Compile, deploy, and run a servlet on Apache Tomcat.

1. Introduction to Servlets

1.1 Definition

A SERVLET is a Java class that runs on a web server and handles HTTP requests from a browser. The word "servlet" comes from "server applet" — a small program that lives on the server. Servlets are the foundation of all Java-based web technologies (JSP, Spring, Struts, Hibernate web apps).

> **Quick definition**
> Servlet = a Java class that listens for HTTP requests, processes them, and sends back HTML, JSON, or other content to the browser. It is Java's answer to PHP.

1.2 Why Servlets?

In Week 10 we used PHP for server-side work. Servlets are the Java alternative. Both do the same job — receive form data, talk to a database, build HTML — but Java offers stronger typing, better tooling, and the rich Java ecosystem (collections, threads, JDBC, security).

- Platform independent — write once, run on any server with Java.
- Multi-threaded — Tomcat handles many requests at once efficiently.
- Robust — full Java type safety catches errors at compile time.
- Mature ecosystem — JDBC for databases, Spring for big apps, Maven/Gradle for builds.
- Used in enterprises — banks, e-commerce, large government systems.

1.3 Servlet Architecture

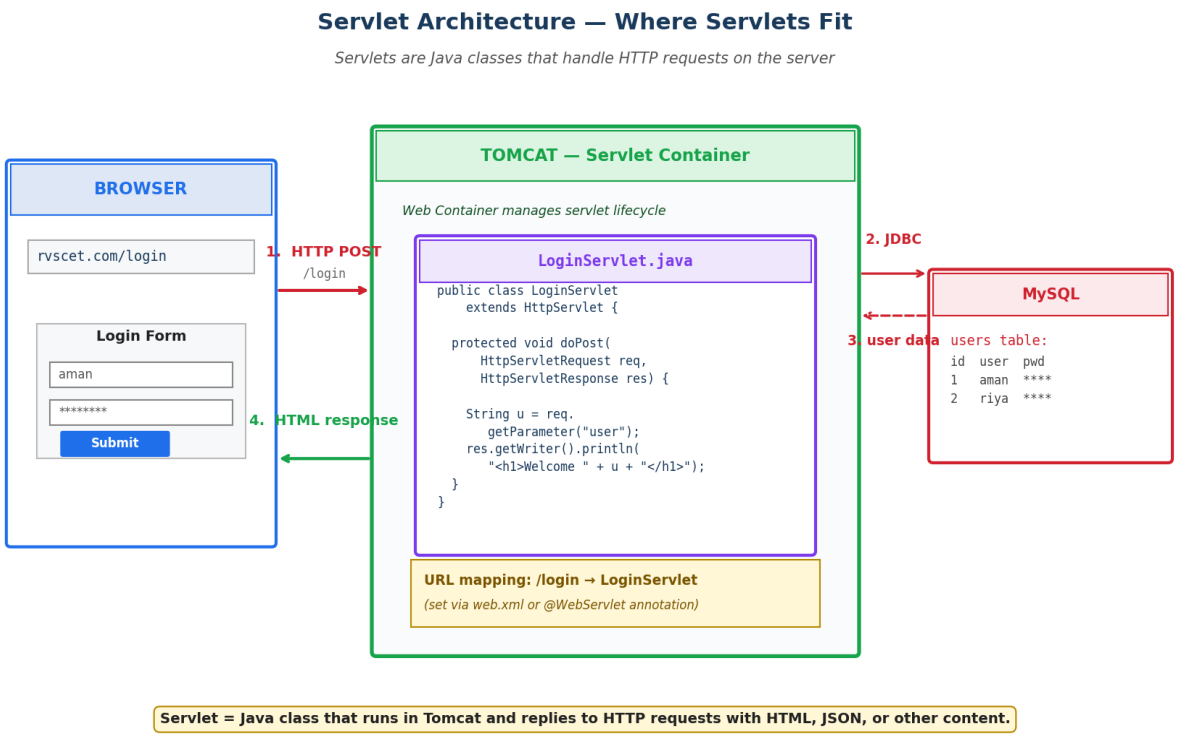


Figure 11.1 — A servlet sits inside Tomcat (the servlet container) and is mapped to a URL like /login.

1.4 Real-World Use Cases

- College admission portals where students apply online (RVSCET admissions, NPTEL exam registration).
- Internet banking — login, fund transfer, statement download.
- E-commerce backends — product search, cart, checkout, payment processing.
- Hostel and library management for institutions.
- REST APIs that power Android apps and React/Angular front-ends.

2. Setting Up the Environment

2.1 What You Need

Tool	Purpose
JDK 8 or later	Java compiler and runtime. Download from oracle.com or use Adoptium.
Apache Tomcat 9+	The servlet container that runs your servlet. Download from tomcat.apache.org.

Tool	Purpose
IDE (optional)	IntelliJ IDEA Community, Eclipse for Enterprise Java, or Apache NetBeans.
servlet-api.jar	The Servlet API library — comes inside Tomcat's lib folder.

2.2 Tomcat Folder Structure

Tomcat directory layout

```
C:\apache-tomcat-9.0.x\
   |-- bin\          <- startup.bat, shutdown.bat
   |-- conf\         <- server.xml (port settings)
   |-- lib\          <- servlet-api.jar lives here
   |-- logs\         <- error logs
   |-- webapps\      <- your apps go HERE
       |-- ROOT\         (default app)
       |-- bca-portal\   (your project)
           |-- index.html
           |-- login.html
           |-- WEB-INF\
               |-- web.xml
               |-- classes\
                   |-- LoginServlet.class
```

2.3 Starting Tomcat

- Open a command prompt in C:\apache-tomcat-9.0.x\bin\.
- Run startup.bat (Windows) or sh startup.sh (Linux/Mac).
- Open the browser and visit http://localhost:8080/ — you should see the Tomcat home page.
- Run shutdown.bat to stop the server.

3. Your First Servlet

3.1 HelloServlet — The Classic First Example

Example 3.1 — HelloServlet.java

```
// File: HelloServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
```

```
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<!DOCTYPE html>");
        out.println("<html><head><title>Hello</title></head>");
        out.println("<body>");
        out.println("<h1>Hello from RVSCET Servlet!</h1>");
        out.println("<p>This page was generated by Java.</p>");
        out.println("</body></html>");
    }
}
```

> ***Output in browser:***
>
> Hello from RVSCET Servlet!
> This page was generated by Java.

3.2 Line-by-Line Explanation

Line	Meaning
import javax.servlet.*	The Servlet API package. Contains base classes.
@WebServlet("/hello")	Annotation that maps this servlet to the URL /hello.
extends HttpServlet	Inherit ready-made HTTP behaviour from the API.
doGet(...)	Method called when the browser sends an HTTP GET request.
request	Object holding all info from the browser (params, headers, cookies).
response	Object you write your reply into (HTML, JSON, headers).
setContentType	Tells the browser what type of content is being returned.
getWriter()	Returns a PrintWriter — like System.out but for the browser.
out.println(...)	Sends a line of HTML back to the browser.

4. Servlet Lifecycle

4.1 The Four Stages

Tomcat is in charge of every servlet — it loads, initialises, calls it, and finally unloads it. This is called the SERVLET LIFECYCLE. Three methods of HttpServlet drive these stages: init(), service() (and its specific versions doGet/doPost), and destroy().

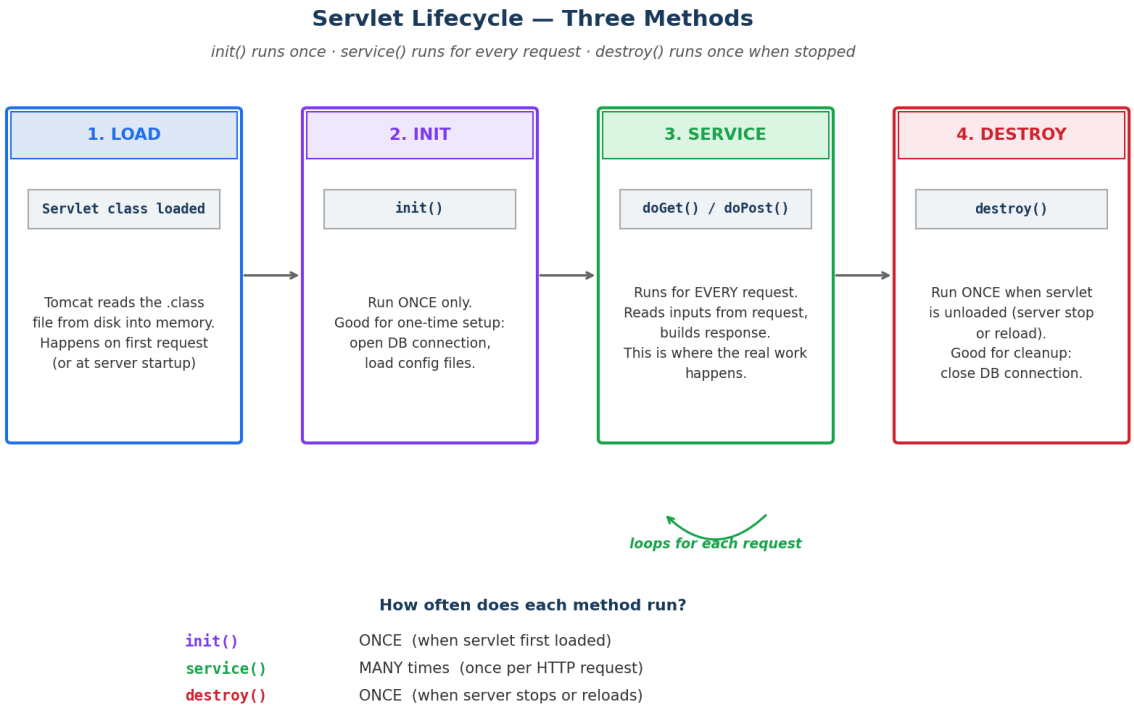


Figure 11.2 — A servlet's lifecycle: load → init → service (loops) → destroy.

4.2 init() — Run Once at Startup

Called by Tomcat exactly once after the servlet is first loaded. Use it for one-time setup like opening a database connection or loading a config file. After init(), the servlet is ready to handle requests.

Example 4.1 — Overriding init()

```
@Override
public void init() throws ServletException {
    System.out.println("Servlet starting up...");
    // open database connection here
    // load configuration files here
}
```

4.3 service() — Run Per Request

Tomcat calls service() every time a new HTTP request arrives. By default, service() looks at the request method and forwards to doGet() or doPost() automatically. We almost never override service() directly — we override doGet() or doPost() instead.

4.4 destroy() — Run Once at Shutdown

Example 4.2 — Overriding destroy()

```
@Override
public void destroy() {
    System.out.println("Servlet shutting down...");
    // close database connection here
}
```

4.5 Putting It Together — A Lifecycle-Aware Servlet

Example 4.3 — Watching the lifecycle in action

```
@WebServlet("/lifecycle")
public class LifecycleServlet extends HttpServlet {

    private int requestCount = 0;

    @Override
    public void init() throws ServletException {
        System.out.println("[INIT] Servlet ready.");
    }

    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
            throws IOException {
        requestCount++;
        res.setContentType("text/html");
        res.getWriter().println(
            "<h2>Request number " + requestCount + "</h2>");
        System.out.println("[SERVICE] Request " + requestCount);
    }

    @Override
    public void destroy() {
        System.out.println("[DESTROY] Total requests: " + requestCount);
    }
}
```

Try this in your lab

1. Deploy the servlet to Tomcat. Watch the Tomcat console.
2. The first time you visit /lifecycle, you'll see [INIT] then [SERVICE] Request 1.
3. Refresh the page — only [SERVICE] Request 2 appears (init() doesn't run again).
4. Stop Tomcat — you'll see [DESTROY] Total requests: 2.

5. HttpServletRequest and HttpServletResponse

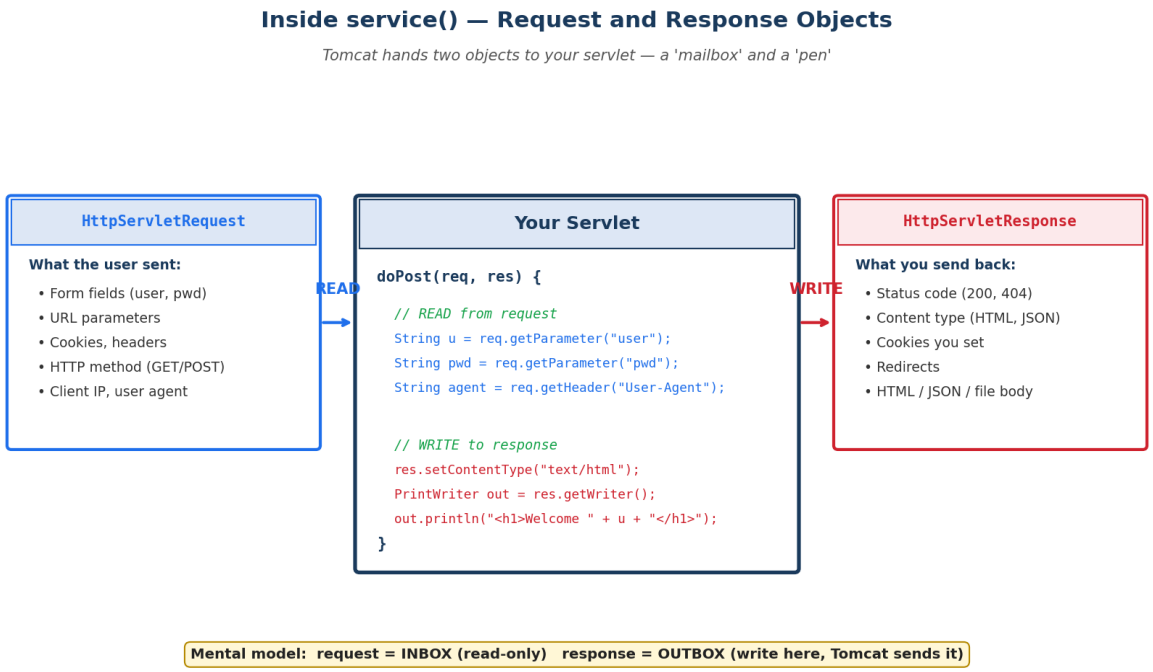


Figure 11.3 — Tomcat hands two objects to your servlet — req for reading inputs, res for writing the reply.

5.1 Reading from the Request

Method	What it returns
request.getParameter("x")	Get a single form field or URL parameter as a String.
request.getParameterValues("x")	Get an array of values (e.g. checkboxes with the same name).
request.getParameterMap()	Get all parameters as a Map<String, String[]>.
request.getMethod()	"GET" or "POST".
request.getRequestURL()	Full URL the user typed.
request.getHeader("Name")	Get a request header (User-Agent, Accept, etc.).
request.getCookies()	Get the array of cookies sent by the browser.
request.getRemoteAddr()	The client's IP address.

5.2 Writing to the Response

Method	What it does
response.setContentType("text/html")	Sets the type of the response body (HTML, JSON, plain text, ...).

Method	What it does
response.getWriter()	Returns a PrintWriter — write HTML/text to the browser.
response.setStatus(404)	Set the HTTP status code (200 OK, 404 Not Found, 500 Error).
response.addCookie(cookie)	Add a cookie to be saved by the browser.
response.sendRedirect("/login")	Tell the browser to load a different URL.
response.setHeader("Name", "Value")	Set a response header.

5.3 Example — A Greeting Servlet

Example 5.1 — GreetServlet

```
@WebServlet("/greet")
public class GreetServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
            throws IOException {

        // 1. Read inputs from request
        String name = req.getParameter("name");
        if (name == null || name.isEmpty()) {
            name = "Guest";
        }

        // 2. Write to response
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html><body>");
        out.println("<h1>Welcome, " + name + "!</h1>");
        out.println("<p>From IP: " + req.getRemoteAddr() + "</p>");
        out.println("</body></html>");
    }
}
```

Visit http://localhost:8080/bca-portal/greet?name=Aman in the browser and see the personalised greeting.

6. doGet() vs doPost()

6.1 When to Use Each

Aspect	doGet()	doPost()
Triggered by	URL bar, link, GET form, refresh.	POST form submission.
Data location	URL parameters (?name=Aman).	Hidden in the request body.
Visible in URL?	Yes — bookmarkable.	No — clean URL.
Best for	Search, filters, page navigation.	Login, registration, payments.
Idempotent?	Yes — repeating it has no side effect.	No — repeating may resubmit a payment.

6.2 Handling Both in One Servlet

Example 6.1 — A login servlet handling GET (show form) and POST (check credentials)

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet {

   // Show the login form when the URL is opened
   @Override
   protected void doGet(HttpServletRequest req,
                        HttpServletResponse res)
         throws IOException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<form action='login' method='post'>");
      out.println("User: <input name='user'><br>");
      out.println("Pwd: <input type='password' name='pwd'><br>");
      out.println("<button>Login</button></form>");
   }

   // Process the login when the form is submitted
   @Override
   protected void doPost(HttpServletRequest req,
                         HttpServletResponse res)
         throws IOException {
      String user = req.getParameter("user");
      String pwd = req.getParameter("pwd");

      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      if ("aman".equals(user) && "bca123".equals(pwd)) {
         out.println("<h1>Welcome, " + user + "!</h1>");
      } else {
         out.println("<h1 style='color:red'>Invalid login</h1>");
      }
   }
}
```

Common pattern

doGet() = show the form (the empty page).

doPost() = process the form data when the user submits.

This pattern is used for nearly every form-based feature in real Java web apps.

7. URL Mapping — web.xml vs Annotations

7.1 Annotation Approach (modern, recommended)

Example 7.1 — @WebServlet annotation

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet { ... }

// Multiple URLs point to the same servlet
@WebServlet(urlPatterns = {"/login", "/signin", "/auth"})
public class LoginServlet extends HttpServlet { ... }
```

7.2 web.xml Approach (older, still common in legacy code)

Example 7.2 — Equivalent web.xml configuration

```
<!-- File: webapps/bca-portal/WEB-INF/web.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         version="4.0">

  <servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>LoginServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/login</url-pattern>
  </servlet-mapping>

</web-app>
```

Aspect	Annotation (@WebServlet)	web.xml
Where	Inside the .java file	In WEB-INF/web.xml
Visibility	Local — close to the code	Centralised — all mappings in one file

Modern?	Yes — recommended since Java EE 6	Older approach
Best for	Most projects today	Large apps with many config-driven mappings

8. Session Tracking with HttpSession

8.1 Why Sessions?

HTTP is stateless — the server forgets you between requests. Once a user logs in, we need a way to remember them on the next page click. Servlets provide HttpSession exactly for this. The browser carries a session ID cookie, and the server keeps the matching session data in memory.

8.2 Working with HttpSession

Method	What it does
request.getSession()	Get the existing session (or create one if none exists).
session.setAttribute("k", v)	Store a value under a key.
session.getAttribute("k")	Read back the value (returns Object, you must cast).
session.removeAttribute("k")	Remove a stored value.
session.invalidate()	End the session — used for logout.
session.setMaxInactiveInterval(n)	Auto-expire after n seconds of inactivity.

8.3 Session Example — Login + Dashboard

Example 8.1a — LoginServlet sets a session attribute

```
// LoginServlet — set session after successful login
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest req,
                          HttpServletResponse res)
            throws IOException {
        String user = req.getParameter("user");
        String pwd = req.getParameter("pwd");

        if ("aman".equals(user) && "bca123".equals(pwd)) {
            HttpSession session = req.getSession();
            session.setAttribute("username", user);
            res.sendRedirect("dashboard");
```

```
        } else {
            res.sendRedirect("login.html?error=1");
        }
    }
}
```

Example 8.1b — DashboardServlet reads from the session

```
// DashboardServlet — protected page
@WebServlet("/dashboard")
public class DashboardServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
            throws IOException {
        HttpSession session = req.getSession(false);  // false = don't create new
        if (session == null || session.getAttribute("username") == null) {
            res.sendRedirect("login.html");
            return;
        }
        String user = (String) session.getAttribute("username");
        res.setContentType("text/html");
        res.getWriter().println("<h1>Welcome, " + user + "!</h1>");
        res.getWriter().println("<a href='logout'>Logout</a>");
    }
}
```

Example 8.1c — LogoutServlet

```
// LogoutServlet — destroy the session
@WebServlet("/logout")
public class LogoutServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
            throws IOException {
        HttpSession session = req.getSession(false);
        if (session != null) session.invalidate();
        res.sendRedirect("login.html");
    }
}
```

8.4 How Sessions Work Behind the Scenes

How HttpSession works under the hood

1. First request: Tomcat creates session (id JSESSIONID=ABC123)
   and sends Set-Cookie header to browser
2. Browser stores: Cookie JSESSIONID=ABC123
3. Next requests: Browser sends Cookie: JSESSIONID=ABC123

```
4. Tomcat looks up:  finds session ABC123 in memory
                     attaches it to request.getSession()
5. Logout:           session.invalidate() removes the data
```

9. Compiling and Deploying a Servlet

9.1 Step-by-Step Without an IDE

- Write your servlet — for example HelloServlet.java.
- Compile it, telling the compiler where to find servlet-api.jar:

Compile command

```
javac -cp "C:\apache-tomcat-9.0.x\lib\servlet-api.jar" HelloServlet.java
```

- Move the resulting HelloServlet.class into the webapp's WEB-INF/classes folder:

Where the .class file goes

```
C:\apache-tomcat-9.0.x\webapps\bca-portal\WEB-INF\classes\HelloServlet.class
```

- Make sure web.xml exists in WEB-INF (or use @WebServlet annotation).
- Start Tomcat. Visit http://localhost:8080/bca-portal/hello.

9.2 Step-by-Step With an IDE

- Open IntelliJ IDEA / Eclipse / NetBeans.
- Create a new project of type "Java Enterprise" or "Web Application".
- Add Apache Tomcat as the runtime / server.
- Create your servlet class — the IDE auto-generates the project structure.
- Click Run / Deploy — the IDE compiles, packages, deploys, and opens the browser.

9.3 WAR Files

For real deployment, packages are usually distributed as a single .war (Web Application Archive) file. A WAR is just a ZIP file containing the WEB-INF folder, classes, JSPs, HTML, and CSS. Drop it into webapps/ and Tomcat unpacks it automatically.

Building a WAR file

```
# Create a WAR file from the project root
jar -cvf bca-portal.war *

# Then drop bca-portal.war into webapps/
```

10. Worked Example — Complete Login Flow

This example combines everything: a login form, a servlet that processes POST data, sessions, a protected dashboard, and a logout link.

Example 10.1a — login.html

```
<!-- File 1: webapps/bca-portal/login.html -->
<!DOCTYPE html>
<html>
<head>
  <title>BCA Portal - Login</title>
  <style>
    body { font-family: Arial; padding: 30px; max-width: 400px; }
    input { width: 100%; padding: 8px; margin: 5px 0 14px; }
    button { padding: 10px 20px; background: #1f6feb;
          color: white; border: none; cursor: pointer; }
  </style>
</head>
<body>
  <h2>BCA Portal - Login</h2>
  <form action="login" method="POST">
    <label>Username:</label>
    <input type="text" name="user" required>

    <label>Password:</label>
    <input type="password" name="pwd" required>

    <button type="submit">Login</button>
  </form>
</body>
</html>
```

Example 10.1b — LoginServlet.java

```
// File 2: LoginServlet.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/login")
public class LoginServlet extends HttpServlet {

  // Hard-coded for the lab; in real apps look up in DB
  private static final String VALID_USER = "aman";
  private static final String VALID_PWD  = "bca123";

  @Override
  protected void doPost(HttpServletRequest req,
                HttpServletResponse res)
      throws IOException, ServletException {
```

```
        String user = req.getParameter("user");
        String pwd  = req.getParameter("pwd");

        if (VALID_USER.equals(user) && VALID_PWD.equals(pwd)) {
            HttpSession session = req.getSession();
            session.setAttribute("username", user);
            session.setAttribute("loginTime", new java.util.Date());
            res.sendRedirect("dashboard");
        } else {
            res.setContentType("text/html");
            res.getWriter().println(
                "<h2 style='color:red'>Invalid credentials.</h2>" +
                "<a href='login.html'>Try again</a>");
        }
    }
}
```

Example 10.1c — DashboardServlet.java

```
// File 3: DashboardServlet.java
@WebServlet("/dashboard")
public class DashboardServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
            throws IOException {

        HttpSession session = req.getSession(false);
        if (session == null || session.getAttribute("username") == null) {
            res.sendRedirect("login.html");
            return;
        }

        String user = (String) session.getAttribute("username");
        Object loginTime = session.getAttribute("loginTime");

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<!DOCTYPE html><html><body>");
        out.println("<h1>Welcome, " + user + "!</h1>");
        out.println("<p>Logged in at: " + loginTime + "</p>");
        out.println("<p>Your IP: " + req.getRemoteAddr() + "</p>");
        out.println("<a href='logout'>Logout</a>");
        out.println("</body></html>");
    }
}
```

Example 10.1d — LogoutServlet.java

```
// File 4: LogoutServlet.java
@WebServlet("/logout")
public class LogoutServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
            throws IOException {
        HttpSession session = req.getSession(false);
        if (session != null) session.invalidate();
        res.sendRedirect("login.html");
    }
}
```

Try this in your lab

1. Save login.html in webapps/bca-portal/.
2. Save the three .java files anywhere convenient.
3. Compile each: javac -cp "%CATALINA_HOME%\lib\servlet-api.jar" *.java.
4. Move the .class files to webapps/bca-portal/WEB-INF/classes/.
5. Restart Tomcat — visit http://localhost:8080/bca-portal/login.html.
6. Log in as aman/bca123 — you go to the dashboard.
7. Try /dashboard directly without logging in — it redirects you back to login.
8. Click Logout — session is destroyed.

11. Summary of Week 11

- A SERVLET is a Java class that handles HTTP requests on a server. It is Java's equivalent of PHP.
- Servlets run inside a SERVLET CONTAINER like Apache Tomcat, which manages their lifecycle.
- The lifecycle has three methods: init() runs once on load, service()/doGet/doPost runs per request, destroy() runs once on unload.
- Use doGet() for GET requests (URL parameters, page loads) and doPost() for POST requests (form submissions).
- HttpServletRequest is the inbox — read parameters, headers, cookies. HttpServletResponse is the outbox — write HTML, set status, redirect.
- URL mappings can be configured with the @WebServlet annotation (modern) or in web.xml (legacy).
- HttpSession tracks logged-in users across requests — store with setAttribute, read with getAttribute, end with invalidate.
- To deploy: place compiled .class files in WEB-INF/classes inside your webapp folder, restart Tomcat.

12. Practice Questions

A. Short Answer (2 marks each)

1. Define a servlet and state in which package the base classes live.

2. What is Apache Tomcat? Where do webapps live in the Tomcat folder structure?

3. List the three lifecycle methods of a servlet and state when each runs.

4. Differentiate between doGet() and doPost() with one use case for each.

5. Write a single line that reads a form field named 'email' from the request.

6. Write the @WebServlet annotation that maps the URL /register to a servlet class.

7. Write the line that destroys a session in a logout servlet.

B. Long Answer (5–10 marks each)

1. Explain the architecture of a servlet-based web application with a neat diagram. Show the role of the browser, Tomcat, the servlet, and the database.

2. Describe the servlet lifecycle in detail with a diagram, naming each method and explaining when and how often each is called. Provide code examples for init() and destroy().

3. Explain HttpServletRequest and HttpServletResponse with at least five methods of each. Write a complete servlet that reads a form input and replies with a personalised greeting.

4. Describe HTTP session tracking in servlets. Write the code for a complete login/dashboard/logout flow using HttpSession.

5. Compare web.xml and the @WebServlet annotation as ways to configure URL mappings. Write equivalent code for both.

C. Lab / Hands-On Tasks

1. Set up Apache Tomcat on your lab computer. Verify the installation by opening http://localhost:8080/.

2. Create a HelloServlet that displays your name, roll number, and the current date and time. Map it to /hello and deploy it.

3. Write a CalculatorServlet with two doPost handlers: one that takes two numbers via a form and returns their sum, and one that returns their product.

4. Build a complete BCA registration form that posts to a RegisterServlet. The servlet should validate inputs and show either a success page or list of errors.

5. Take the worked example login system and extend it: maintain a 'visit count' in the session, and display it on the dashboard. Each refresh should increase the count.

WEEK 12 — JSP AND MVC

Course Outcome: CO4 | Topics: JSP intro, scriptlets, expressions, directives, JSTL, JavaBeans, MVC pattern

Learning Objectives

By the end of this week, students will be able to:

- Define JSP and explain how it relates to a servlet.
- Describe the JSP page-translation lifecycle (.jsp → .java → .class → execution).
- Use JSP scriptlets, expressions, and declarations to embed Java in HTML.
- Use JSP directives — page, include, taglib.

- Use the implicit objects (request, response, out, session, application).
- Use the JSP Standard Tag Library (JSTL) to remove scriptlets from views.
- Define a JavaBean and use it to carry data between the Servlet and JSP.
- Apply the Model-View-Controller (MVC) pattern using Servlet + JavaBean + JSP.

1. Introduction to JSP

1.1 Definition

JSP stands for JAVA SERVER PAGES. A JSP file is essentially an HTML file with small pieces of Java code embedded inside it. When the browser requests the page, Tomcat secretly turns the JSP into a servlet, compiles it, and runs it — but the developer writes mostly HTML, which is much more pleasant than typing out.println("<h1>...</h1>") for every line.

> **Quick definition**
> JSP = HTML with Java code mixed in. The file ends with .jsp, not .java. Tomcat translates each .jsp into a servlet behind the scenes.

1.2 Why JSP?

In Week 11 we used Servlets for everything — including printing HTML. That works, but the HTML gets buried inside Java string literals and becomes hard to read. JSP flips the perspective: write HTML normally, drop in Java only where you need dynamic content.

- Cleaner code — most of the file is plain HTML, with Java only in small bursts.
- Designer-friendly — front-end developers can edit the HTML/CSS without touching Java.
- Same speed as Servlets — JSPs are compiled to servlets, so the runtime is identical.
- Implicit objects — request, response, session, out are ready to use without setup.
- Tag libraries (JSTL) can replace Java code with simple HTML-like tags.

1.3 How a JSP Becomes a Servlet

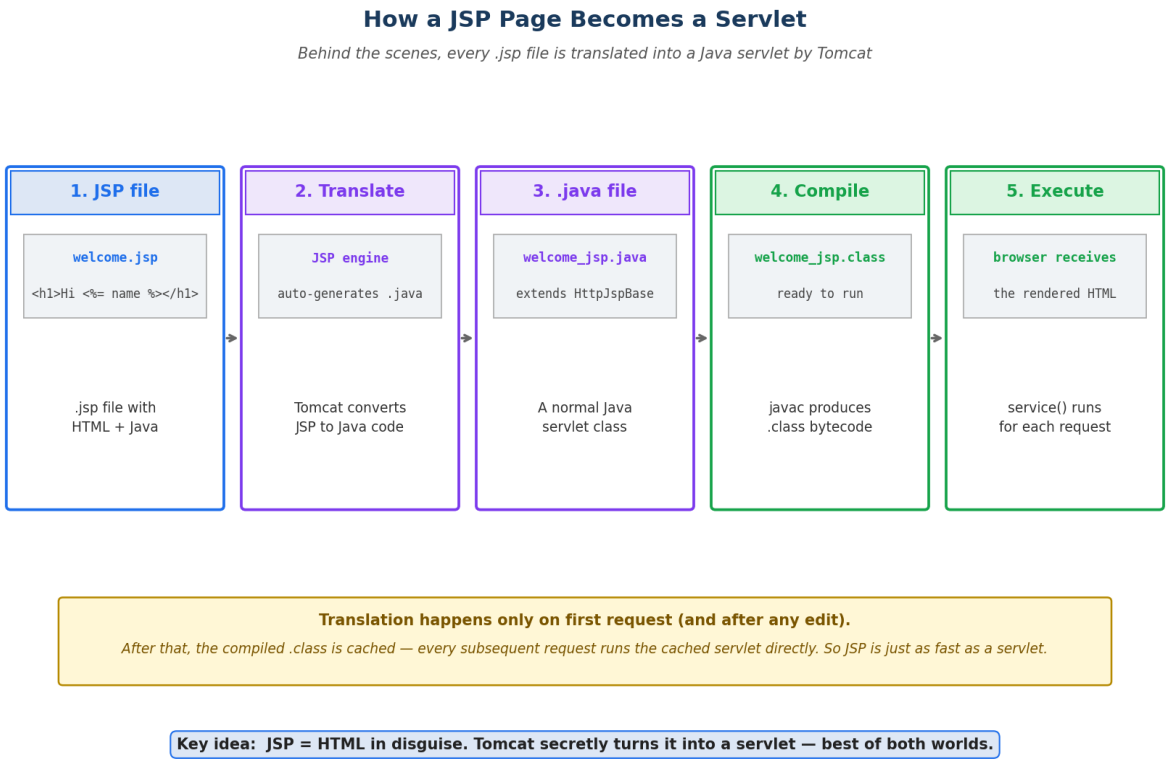


Figure 12.1 — A .jsp file is translated into a .java servlet, compiled to .class, and then executed.

1.4 JSP vs Servlet — Side by Side

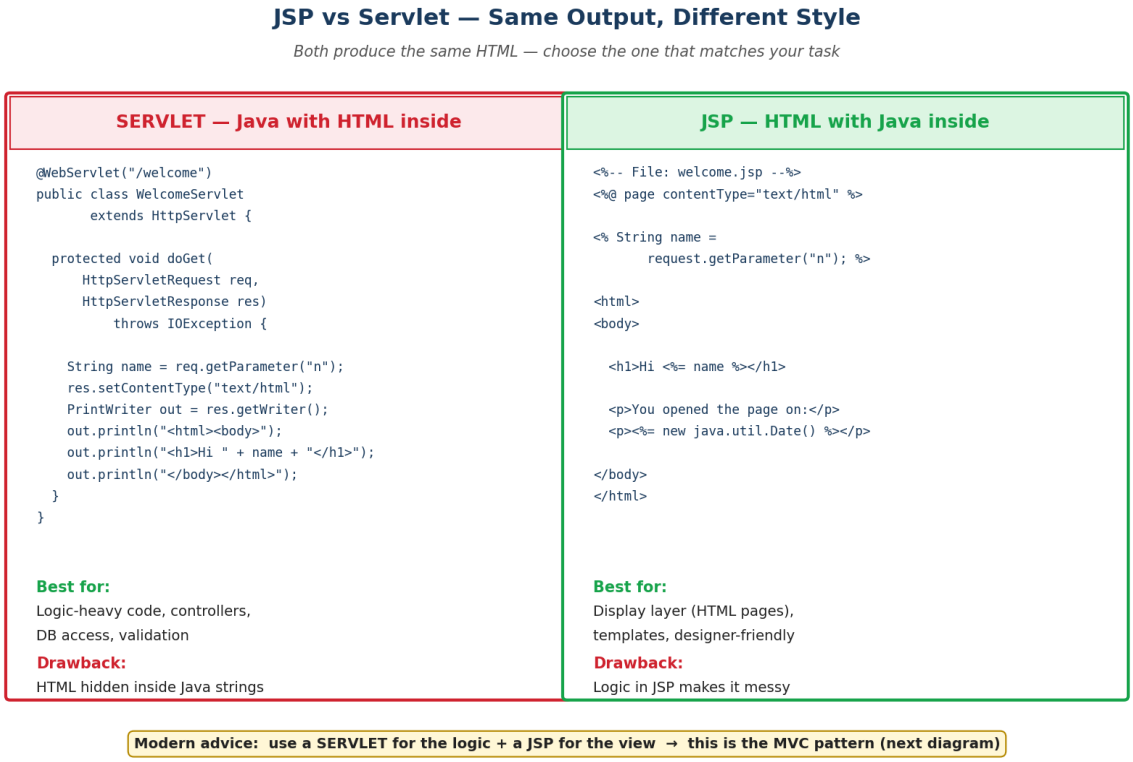


Figure 12.2 — Both files produce the same HTML, but the servlet hides HTML inside Java strings while the JSP keeps HTML visible and Java small.

Aspect	Servlet	JSP
File type	.java	.jsp
Style	Java with HTML inside Strings	HTML with Java inside <% ... %>
Best for	Logic-heavy code, controllers, APIs	Display layer (the View)
Compiled?	Yes (you compile manually)	Yes (Tomcat compiles for you)
Edit cycle	Edit → compile → restart Tomcat	Edit → just refresh the browser

1.5 Real-World Use Cases

- Result publishing pages — JSP renders a student's marks pulled from the database.
- Online shopping product pages — JSP displays each product with photo, price, reviews.
- Dashboards — JSPs show charts and tables generated server-side.
- Admin panels for college portals — list of registered students, fee defaulters, etc.
- Email templates and printable reports rendered as HTML by JSP.

2. Your First JSP Page

2.1 Setup and Folder Structure

JSP files live alongside your other web files inside the webapp folder — they do NOT go in WEB-INF/classes (that is only for compiled Java).

Where JSP files live in a Tomcat webapp

```
C:\apache-tomcat-9.0.x\webapps\bca-portal\
   |-- index.html
   |-- welcome.jsp          <-- JSP files go here
   |-- result.jsp
   |-- WEB-INF\
       |-- web.xml
       |-- classes\         (servlets only)
       |-- lib\             (JAR libraries)
```

2.2 Hello, JSP!

Example 2.1 — welcome.jsp

```
<%-- File: welcome.jsp --%>
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Welcome</title>
</head>
<body>
  <h1>Hello from JSP at RVSCET!</h1>
  <p>The current date and time is:
    <%= new java.util.Date() %></p>
</body>
</html>
```

> ***Output in browser:***

Hello from JSP at RVSCET!
The current date and time is: Tue Apr 28 10:21:00 IST 2026

Try this in your lab

1. Save the file as welcome.jsp inside webapps/bca-portal/.
2. Start Tomcat (startup.bat).
3. Open http://localhost:8080/bca-portal/welcome.jsp in Chrome.
4. Refresh — the time updates with each request, proving Java is running on every visit.

3. JSP Scripting Elements

JSP gives us THREE ways to embed Java code in a page. Each has a different syntax and a different purpose.

Element	Syntax	Used for
Scriptlet	<% ... %>	Multiple lines of Java code (statements).
Expression	<%= ... %>	A single value to be printed in the HTML.
Declaration	<%! ... %>	Method or variable definitions for the page.

3.1 Scriptlets — <% ... %>

Scriptlets contain ordinary Java statements — variable declarations, if/else, loops, method calls. The Java code is run on the server when the page is requested.

Example 3.1 — Scriptlet with a for loop

```
<%@ page contentType="text/html" %>
<html>
```

```
<body>
  <h1>Multiplication Table</h1>

  <% int n = 5; %>

  <table border="1" cellpadding="6">
  <% for (int i = 1; i <= 10; i++) { %>
    <tr>
      <td><%= n %></td>
      <td>x</td>
      <td><%= i %></td>
      <td>=</td>
      <td><%= n * i %></td>
    </tr>
  <% } %>
  </table>
</body>
</html>
```

> ***Output in browser:***

Multiplication Table
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
...
5 x 10 = 50

3.2 Expressions — <%= ... %>

Expressions evaluate a single Java expression and print the result directly into the HTML output. Notice there is NO semicolon — this is just an expression, not a statement.

Example 3.2 — Five JSP expressions

```
<p>Server time: <%= new java.util.Date() %></p>
<p>2 + 3 = <%= 2 + 3 %></p>
<p>Welcome, <%= request.getParameter("name") %>!</p>
<p>Your IP: <%= request.getRemoteAddr() %></p>
<p>Random: <%= Math.random() %></p>
```

3.3 Declarations — <%! ... %>

Declarations let you define instance variables and methods that belong to the generated servlet class itself, not just to the service() method. Use them sparingly — most modern JSP pages avoid declarations.

Example 3.3 — Declaring a variable and a helper method

```
<%!
```

```
    private int counter = 0;

    private String getGreeting(int hour) {
        if (hour < 12) return "Good Morning";
        else if (hour < 17) return "Good Afternoon";
        else return "Good Evening";
    }
%>

<p>Visits so far: <%= ++counter %></p>
<p><%= getGreeting(java.time.LocalTime.now().getHour()) %>!</p>
```

3.4 Comments

Example 3.4 — Three styles of comments in JSP

```
<%-- JSP comment — NOT visible in the output HTML --%>

<!-- HTML comment — visible if user views page source -->

<%
    // Java single-line comment
    /* Java multi-line
       comment */
%>
```

4. JSP Directives

DIRECTIVES are instructions to the JSP engine about how to translate the page. They start with <%@ and end with %>. There are three of them: page, include, and taglib.

4.1 page Directive

Sets page-wide settings: content type, character encoding, imports, error handling, session usage.

Example 4.1 — Common page directive attributes

```
<%@ page contentType="text/html" pageEncoding="UTF-8" %>
<%@ page import="java.util.*, java.text.*" %>
<%@ page errorPage="error.jsp" %>
<%@ page session="true" %>
<%@ page isErrorPage="false" %>
```

Attribute	Purpose
contentType	MIME type and encoding of the response (e.g. text/html;charset=UTF-8).
pageEncoding	Character set used to read the JSP file itself.

import	Comma-separated Java classes/packages to import (replaces import in scriptlet).
session	true (default) means request.getSession() is auto-available.
errorPage	URL of a JSP that handles uncaught exceptions thrown by this page.
isErrorPage	true marks this JSP as an error handler so it can use the implicit 'exception' object.
buffer	Size of the output buffer (default 8kb).

4.2 include Directive

The include directive copies another file's content into the current JSP at translation time. Useful for shared headers, footers, and navigation bars.

Example 4.2a — header.jsp

```
<%-- File: header.jsp --%>
<header style="background:#1a3a5c;color:white;padding:10px">
  <h1>RVSCET — BCA Portal</h1>
</header>
```

Example 4.2b — home.jsp uses include

```
<%-- File: home.jsp --%>
<%@ page contentType="text/html" %>
<html><body>
  <%@ include file="header.jsp" %>

  <main>
    <h2>Welcome to BCA</h2>
    <p>This is the home page.</p>
  </main>

  <%@ include file="footer.jsp" %>
</body></html>
```

4.3 taglib Directive

Imports a tag library (like JSTL) into the page so its custom tags can be used. We will see this in section 6.

Example 4.3 — Importing JSTL core tags with prefix c

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

5. JSP Implicit Objects

JSP pre-creates several objects that are READY TO USE inside any JSP without declaration. These are called IMPLICIT OBJECTS. The most useful ones are listed below.

Implicit Object	What it represents
request	HttpServletRequest — read parameters, headers, cookies.
response	HttpServletResponse — set status, content type, redirect.
out	JspWriter — write text to the output (rarely used; <%= %> is better).
session	HttpSession — store user-specific data across requests.
application	ServletContext — share data across the WHOLE webapp.
pageContext	Bag of all the above; useful in JSTL.
config	ServletConfig — init parameters for the servlet generated from the JSP.
page	Refers to 'this' (the generated servlet).
exception	The Throwable — only available in error pages (isErrorPage="true").

5.1 Using request and out

Example 5.1 — Using request to read a query parameter

```
<%-- File: greet.jsp --%>
<%@ page contentType="text/html" %>
<html><body>

<% String name = request.getParameter("name"); %>

<% if (name == null || name.isEmpty()) { %>
   <h1>Hello, Guest!</h1>
<% } else { %>
   <h1>Hello, <%= name %>!</h1>
   <p>Your IP: <%= request.getRemoteAddr() %></p>
<% } %>

</body></html>
```

Visit greet.jsp?name=Aman in the browser to see the personalised greeting.

5.2 Using session

Example 5.2 — A simple per-session visit counter

```
<%-- counter.jsp --%>
<%@ page contentType="text/html" %>
```

```
<%
   Integer count = (Integer) session.getAttribute("visits");
   if (count == null) count = 0;
   count++;
   session.setAttribute("visits", count);
%>
<html><body>
   <p>You have visited this page <%= count %> time(s) this session.</p>
</body></html>
```

6. JSP Standard Tag Library (JSTL)

6.1 Why JSTL?

Scriptlets (the <% %> blocks) make a JSP messy: HTML and Java mixed together, hard for designers to read. JSTL — the JSP Standard Tag Library — replaces common scriptlet tasks (loops, conditionals, output) with clean HTML-like custom tags.

6.2 Adding JSTL to a Project

- Download jstl-1.2.jar (or the modern Jakarta version) and put it in WEB-INF/lib/.
- In every JSP that uses JSTL, add the taglib directive at the top:

JSTL core tags taglib directive

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

6.3 Common JSTL Core Tags

Tag	Purpose
<c:out value="...">	Print a value (auto-escapes HTML, safer than <%= %>).
<c:set var="x" value="...">	Set a variable (in the page or session scope).
<c:if test="${cond}">	Run the body if the condition is true.
<c:choose>...<c:when>	Multi-branch if-else if-else.
<c:forEach var="i" items="${list}">	Loop over a collection or array.
<c:forEach begin="1" end="5">	Loop a fixed number of times.
<c:redirect url="..."/>	Send the user to a different URL.
<c:url value="..."/>	Build a URL safely (encodes parameters).

6.4 JSTL Examples

Example 6.1 — JSTL replacing scriptlets

```
<%@ page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%-- Set a variable --%>
<c:set var="name" value="Aman Kumar"/>

<%-- Print it (safer than <%= %>) --%>
<h1>Welcome, <c:out value="${name}"/>!</h1>

<%-- Conditional --%>
<c:if test="${not empty name}">
   <p>Your name is <c:out value="${name}"/>.</p>
</c:if>

<%-- Counted loop --%>
<ul>
<c:forEach var="i" begin="1" end="5">
   <li>Item <c:out value="${i}"/></li>
</c:forEach>
</ul>
```

6.5 Looping Over a Collection

When the controller (servlet) puts a List<Student> into the request, the JSP can loop over it without any Java code:

Example 6.2 — Looping over a List of Student objects

```
<%-- The Servlet earlier did:
      request.setAttribute("students", listOfStudents);
      request.getRequestDispatcher("list.jsp").forward(req,res);
--%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<table border="1">
   <tr><th>Name</th><th>Marks</th></tr>
   <c:forEach var="s" items="${students}">
      <tr>
         <td><c:out value="${s.name}"/></td>
         <td><c:out value="${s.marks}"/></td>
      </tr>
   </c:forEach>
</table>
```

EL — Expression Language

The ${...} syntax inside JSTL is called the JSP Expression Language. It is a short, safe way to read attributes from request, session, or page scope. ${name} looks first in the page, then request, then session, then application — automatically.

7. JavaBeans

7.1 What is a JavaBean?

A JavaBean is just a regular Java class that follows three simple rules. Beans are used to carry structured data between layers of an MVC application — for example, a Servlet creates a Student bean from form data and passes it to a JSP for display.

- Has a public, no-argument constructor.
- All fields are private.
- Each field has a public getter (getX) and setter (setX).

7.2 A Student JavaBean

Example 7.1 — Student.java JavaBean

```
// File: Student.java
package com.rvscet.beans;

public class Student {
   private String name;
   private int    marks;
   private String course;

   // No-argument constructor (required)
   public Student() {
   }

   // Convenience constructor
   public Student(String name, int marks, String course) {
      this.name = name;
      this.marks = marks;
      this.course = course;
   }

   // Getters and Setters
   public String getName()       { return name; }
   public void setName(String n)  { this.name = n; }

   public int getMarks()         { return marks; }
   public void setMarks(int m)    { this.marks = m; }
```

```
    public String getCourse()       { return course; }
    public void setCourse(String c) { this.course = c; }

    public String getGrade() {
        if (marks >= 90) return "A+";
        else if (marks >= 80) return "A";
        else if (marks >= 70) return "B";
        else if (marks >= 40) return "C";
        else return "F";
    }
}
```

7.3 Using a JavaBean in a JSP

Example 7.2 — Using <jsp:useBean> in result.jsp

```
<%@ page contentType="text/html" %>
<%@ page import="com.rvscet.beans.Student" %>

<jsp:useBean id="student" class="com.rvscet.beans.Student" scope="page"/>
<jsp:setProperty name="student" property="name"   value="Aman"/>
<jsp:setProperty name="student" property="marks"  value="85"/>
<jsp:setProperty name="student" property="course" value="BCA"/>

<html><body>
    <h1>Result Card</h1>
    <p>Name:   <jsp:getProperty name="student" property="name"/></p>
    <p>Marks:  <jsp:getProperty name="student" property="marks"/></p>
    <p>Course: <jsp:getProperty name="student" property="course"/></p>
    <p>Grade:  <jsp:getProperty name="student" property="grade"/></p>
</body></html>
```

Tag / Attribute	Meaning
<jsp:useBean>	Creates or finds a bean instance in a given scope.
<jsp:setProperty>	Calls a setter on the bean (setName, setMarks, ...).
<jsp:getProperty>	Calls a getter and prints the result in the page.
scope="page"	Bean lives only for this JSP request.
scope="request"	Bean lives for the current request (good for forwarding).
scope="session"	Bean lives for the user's whole session (good for cart, login).
scope="application"	Bean shared across the whole webapp.

8. MVC — Model, View, Controller

8.1 What is MVC?

MVC is a software design pattern that splits a web app into three layers, each with one clear responsibility. The pattern was invented in the 1970s and has since become the standard way to organise web apps in every major framework — Java (Servlet+JSP, Spring), PHP (Laravel), Ruby (Rails), Python (Django), and so on.

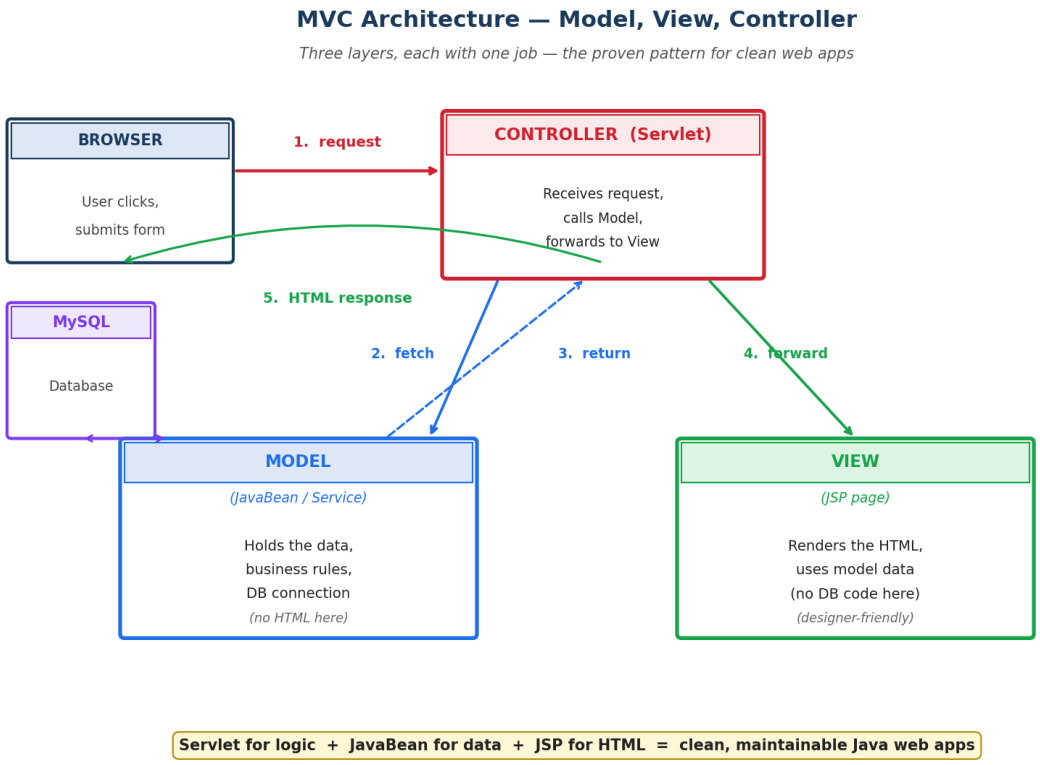


Figure 12.3 — The classic MVC flow: browser → controller → model (DB) → controller → view → browser.

Layer	Role
MODEL	JavaBean / Service / DAO. Holds data and business rules. NO HTML. Examples: Student.java, StudentDAO.java.
VIEW	JSP page. Renders HTML using model data. NO database code. Examples: list.jsp, result.jsp.
CONTROLLER	Servlet. Receives HTTP requests, calls Model, forwards to View. Examples: StudentServlet.java.

8.2 Why Use MVC?

- Separation of concerns — each file has ONE job, not three.
- Maintainability — change the look (View) without touching logic; change DB schema (Model) without touching HTML.
- Team work — designers edit JSPs while developers edit servlets.

- Testability — Model and Controller can be tested without a browser.
- Industry standard — every major Java framework (Spring, Struts) is built on MVC.

8.3 Forwarding from a Servlet to a JSP

The bridge between Controller and View is RequestDispatcher.forward(). The servlet attaches data using request.setAttribute() and forwards the request to a JSP, which then reads the data and renders the HTML.

Example 8.1 — Controller passes a bean to a JSP

```
// In the servlet (Controller)
Student s = new Student("Aman", 85, "BCA");
request.setAttribute("student", s);
request.getRequestDispatcher("result.jsp").forward(request, response);
```

Example 8.2 — JSP reads the bean using EL

```
<%-- result.jsp (View) --%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html><body>
  <h1>Result Card</h1>
  <p>Name:  <c:out value="${student.name}"/></p>
  <p>Marks:  <c:out value="${student.marks}"/></p>
  <p>Grade:  <c:out value="${student.grade}"/></p>
</body></html>
```

9. Worked Example — A Full MVC Result Card App

Let's combine everything: a JSP form, a JavaBean to carry the data, a Servlet acting as Controller, and a JSP View showing the result card. This example uses all three weeks of server-side material — Forms (Week 4), Servlets (Week 11), and JSP (Week 12) — bound together by MVC.

9.1 File 1 — index.jsp (the form)

Example 9.1 — index.jsp

```
<%@ page contentType="text/html" %>
<!DOCTYPE html>
<html>
<head>
  <title>BCA Result Lookup</title>
  <style>
    body { font-family: Arial; padding: 30px; max-width: 400px; }
    input { width: 100%; padding: 8px; margin: 5px 0 14px; }
    button { padding: 10px 20px; background: #1f6feb;
          color: white; border: none; cursor: pointer; }
  </style>
</head>
<body>
```

```
    <h1>BCA Result Lookup</h1>
    <form action="result" method="POST">
        <label>Name:</label>
        <input type="text" name="name" required>

        <label>Marks (0-100):</label>
        <input type="number" name="marks" min="0" max="100" required>

        <label>Course:</label>
        <input type="text" name="course" value="BCA" required>

        <button type="submit">Show Result</button>
    </form>
</body>
</html>
```

9.2 File 2 — Student.java (the Model / JavaBean)

Example 9.2 — Student.java

```
package com.rvscet.beans;

public class Student {
    private String name;
    private int marks;
    private String course;

    public Student() {}

    public Student(String n, int m, String c) {
        this.name = n; this.marks = m; this.course = c;
    }

    public String getName()        { return name; }
    public void setName(String n)  { this.name = n; }
    public int getMarks()          { return marks; }
    public void setMarks(int m)    { this.marks = m; }
    public String getCourse()      { return course; }
    public void setCourse(String c) { this.course = c; }

    public String getGrade() {
        if (marks >= 90) return "A+";
        if (marks >= 80) return "A";
        if (marks >= 70) return "B";
        if (marks >= 40) return "C";
        return "F";
    }
```

```
    public String getStatus() {
        return marks >= 40 ? "PASS" : "FAIL";
    }
}
```

9.3 File 3 — ResultServlet.java (the Controller)

Example 9.3 — ResultServlet.java

```
package com.rvscet.controllers;

import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
import com.rvscet.beans.Student;

@WebServlet("/result")
public class ResultServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req,
                          HttpServletResponse res)
            throws ServletException, IOException {

        // 1. Read form data
        String name   = req.getParameter("name");
        int    marks  = Integer.parseInt(req.getParameter("marks"));
        String course = req.getParameter("course");

        // 2. Build the Model
        Student s = new Student(name, marks, course);

        // 3. Attach to request and forward to View
        req.setAttribute("student", s);
        req.getRequestDispatcher("result.jsp").forward(req, res);
    }
}
```

9.4 File 4 — result.jsp (the View)

Example 9.4 — result.jsp

```
<%@ page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
    <title>Result Card</title>
```

```
    <style>
      body { font-family: Arial; padding: 30px; max-width: 500px; }
      .card { border: 2px solid #1f6feb; border-radius: 8px;
              padding: 20px; background: #f4f6f8; }
      .pass  { color: #16a34a; font-weight: bold; }
      .fail  { color: #cf222e; font-weight: bold; }
    </style>
  </head>
  <body>
    <h1>BCA — Result Card</h1>

    <div class="card">
      <p><strong>Name:</strong>   <c:out value="${student.name}"/></p>
      <p><strong>Course:</strong> <c:out value="${student.course}"/></p>
      <p><strong>Marks:</strong>  <c:out value="${student.marks}"/> / 100</p>
      <p><strong>Grade:</strong>  <c:out value="${student.grade}"/></p>
      <p><strong>Status:</strong>
        <c:choose>
          <c:when test="${student.status == 'PASS'}">
            <span class="pass">PASS</span>
          </c:when>
          <c:otherwise>
            <span class="fail">FAIL</span>
          </c:otherwise>
        </c:choose>
      </p>
    </div>

    <p><a href="index.jsp">Try another student</a></p>
  </body>
  </html>
```

Try this in your lab

1. Create the structure under webapps/result-app/.
2. Compile Student.java and ResultServlet.java with javac.
3. Place them under WEB-INF/classes/com/rvscet/beans/ and WEB-INF/classes/com/rvscet/controllers/.
4. Place index.jsp and result.jsp directly inside webapps/result-app/.
5. Place jstl-1.2.jar inside WEB-INF/lib/.
6. Restart Tomcat. Visit http://localhost:8080/result-app/.
7. Enter "Aman / 85 / BCA" and submit — see the styled result card with grade B and PASS in green.

9.5 Folder Structure of the Final App

Final layout

```
webapps/result-app/
  |-- index.jsp              <-- form (View)
  |-- result.jsp             <-- result page (View)
  |-- WEB-INF/
      |-- lib/
      |   |-- jstl-1.2.jar
      |-- classes/
          |-- com/rvscet/beans/Student.class       <-- Model
          |-- com/rvscet/controllers/ResultServlet.class  <-- Controller
```

10. Summary of Week 12

- JSP (Java Server Pages) lets you write HTML pages with small pieces of Java embedded — Tomcat translates them into servlets behind the scenes.
- Three scripting elements: scriptlets <% %>, expressions <%= %>, declarations <%! %>.
- Three directives: page (settings + imports), include (combine files), taglib (load tag libraries).
- Implicit objects (request, response, session, application, out, pageContext, ...) are ready to use without setup.
- JSTL replaces scriptlets with clean tags: <c:out>, <c:if>, <c:choose>, <c:forEach>.
- EL (${...}) is a short, safe way to read attributes inside JSTL and JSP.
- A JavaBean is a class with private fields, public getters/setters, and a no-arg constructor — used to carry structured data.
- MVC pattern: Servlet (Controller) + JavaBean/DAO (Model) + JSP (View) — the standard architecture for Java web apps.
- Controller forwards data to View using request.setAttribute() and request.getRequestDispatcher().forward().

11. Practice Questions

A. Short Answer (2 marks each)

1. Define JSP and explain how it differs from a servlet.

2. Write the syntax of the three JSP scripting elements with one example of each.

3. What is the purpose of the page directive? List any three of its attributes.

4. List any five JSP implicit objects with their purpose.

5. What is JSTL? Write the taglib directive needed to import its core library.

6. What is a JavaBean? List its three rules.

7. Explain in two lines what MVC stands for and what each layer does.

B. Long Answer (5–10 marks each)

1. Explain the JSP lifecycle with a neat diagram. Show how a .jsp file is converted into a servlet and executed.

2. Compare JSP and Servlets with code examples for both. Write the same 'Hello World' page in each style and explain the trade-offs.

3. Describe at least seven JSP implicit objects with their roles. Provide a scriptlet example using request and session.

4. Discuss JSTL with examples of <c:if>, <c:choose>, and <c:forEach>. Why is JSTL preferred over scriptlets in modern JSP?

5. Explain the MVC architecture with a diagram. Walk through what each layer does using a sample 'Result Card' application.

C. Lab / Hands-On Tasks

1. Create a JSP page that shows the current server date, time, and your IP address. Use only expressions <%= %>.

2. Build a JSP that asks the user for their name and age via a form, then prints whether they are eligible to vote.

3. Convert any earlier servlet (e.g. the Week 11 GreetServlet) into a JSP page. Compare the two — which is easier to read?

4. Write a JavaBean named Employee with name, salary, and getTax() method (10% if salary > 50000). Use <jsp:useBean> in a JSP to display an employee's tax.

5. Build a complete MVC mini-app: a form (View 1) → ResultServlet (Controller) → Student JavaBean (Model) → ResultPage JSP (View 2). Display the grade and status using JSTL.

WEEK 13 — MYSQL, JDBC, XML & WEB SERVICES

Course Outcome: CO5 | Topics: SQL basics, JDBC, prepared statements, XML, REST web services

Learning Objectives

By the end of this week, students will be able to:

- Explain what a database is and write basic MySQL CRUD queries (CREATE, READ, UPDATE, DELETE).
- Use phpMyAdmin or the MySQL command-line to create databases and tables.
- Describe JDBC architecture and the five steps of a JDBC program.
- Connect a Java application to MySQL using JDBC and run SELECT, INSERT, UPDATE, DELETE.
- Use PreparedStatement to write safe queries that prevent SQL injection.
- Read and write XML documents and explain when XML is used.
- Define a web service and explain the difference between SOAP and REST.
- Build a small REST endpoint using a servlet that returns JSON.

1. Introduction to Databases and MySQL

1.1 Why a Database?

Until now, the data in our PHP and Servlet examples disappeared the moment Tomcat or Apache stopped — variables, sessions, and arrays only live in memory. A real web app needs PERSISTENT STORAGE: a place where students, marks, products, and orders survive a server restart and can be searched, updated, and deleted. A DATABASE is exactly that — an organised store of data that lives on disk and answers queries efficiently.

1.2 What is MySQL?

MySQL is the world's most popular open-source RELATIONAL DATABASE. It stores data in tables made of rows and columns, like a spreadsheet — but with strict types, indexes for fast search, and a powerful query language called SQL (Structured Query Language). MySQL is free, runs on every platform, and powers WordPress, Wikipedia, YouTube, and countless college projects.

Table example
Imagine a paper register where each row is one student and the columns are id, name, course, marks. MySQL is that register, only on a computer — much faster, with thousands of rows.

1.3 SQL — Four Basic Operations (CRUD)

Operation	SQL Keyword	Purpose
Create	INSERT	Add a new row to a table.
Read	SELECT	Fetch one or more rows that match a condition.
Update	UPDATE	Change values in existing rows.
Delete	DELETE	Remove rows from a table.

1.4 Creating a Database and Table

Example 1.1 — SQL to create a database and table

```
-- Create a new database
CREATE DATABASE bca_portal;

-- Switch to it
USE bca_portal;

-- Create a 'students' table
CREATE TABLE students (
    id     INT AUTO_INCREMENT PRIMARY KEY,
    name   VARCHAR(50) NOT NULL,
    course VARCHAR(20),
    marks  INT,
    email  VARCHAR(100) UNIQUE
);
```

SQL Item	Meaning
INT	Whole numbers.
VARCHAR(50)	Variable-length text up to 50 characters.

TEXT	Long text — paragraphs.
DATE / DATETIME	Date and time values.
BOOLEAN	true or false (stored as 0/1).
PRIMARY KEY	Unique identifier for each row — usually 'id'.
AUTO_INCREMENT	Auto-generates next id (1, 2, 3, ...).
NOT NULL	Field cannot be empty.
UNIQUE	No two rows can have the same value.

1.5 CRUD Examples

Example 1.2 — Basic CRUD queries

```
-- INSERT (Create)
INSERT INTO students (name, course, marks, email)
VALUES ('Aman Kumar',  'BCA', 85, 'aman@rvscet.com');

INSERT INTO students (name, course, marks, email)
VALUES ('Riya Singh',  'BCA', 92, 'riya@rvscet.com'),
       ('Vivek Verma', 'BCA', 67, 'vivek@rvscet.com');

-- SELECT (Read)
SELECT * FROM students;                      -- everyone
SELECT name, marks FROM students;            -- two columns
SELECT * FROM students WHERE marks >= 80;    -- top scorers
SELECT * FROM students WHERE course = 'BCA'
       ORDER BY marks DESC;                  -- sorted by marks

-- UPDATE
UPDATE students SET marks = 90 WHERE id = 1;
UPDATE students SET course = 'MCA' WHERE marks > 90;

-- DELETE
DELETE FROM students WHERE id = 3;
DELETE FROM students WHERE marks < 40;
```

1.6 Setting Up MySQL with XAMPP

- Open the XAMPP Control Panel and click START next to MySQL.
- Visit http://localhost/phpmyadmin/ — a friendly web UI for MySQL.
- Click Databases → enter 'bca_portal' → Create.
- Click the new database → SQL tab → paste the CREATE TABLE statement → Go.
- Click the table name → Insert tab to add rows manually, or Browse to view existing rows.

2. JDBC — Java Database Connectivity

2.1 What is JDBC?

JDBC stands for JAVA DATABASE CONNECTIVITY. It is the standard Java API that lets a Java program connect to any relational database, run SQL queries, and read the results. Just as ODBC connects Windows programs to databases, JDBC connects Java programs.

> **Quick definition**
> JDBC = a Java library (java.sql.*) + a database-specific DRIVER (a JAR file). Together they let your Java code run SQL on MySQL, Oracle, SQL Server, PostgreSQL, and others without changing the code much.

2.2 JDBC Architecture

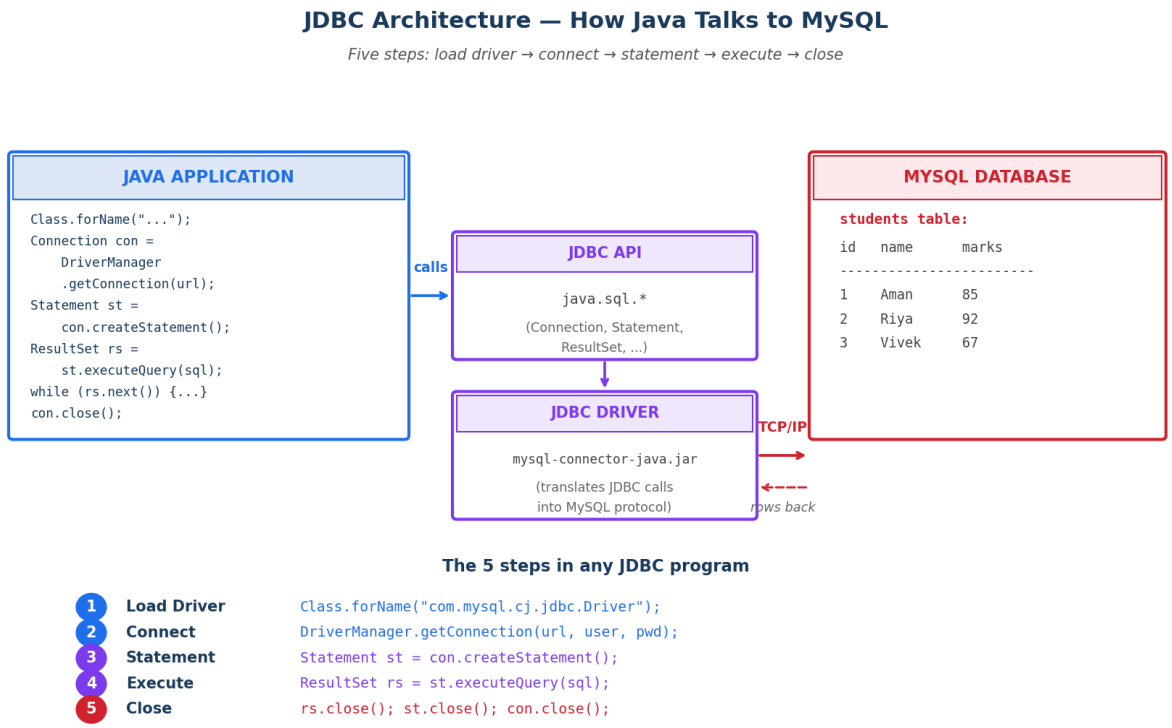


Figure 13.1 — JDBC sits between your Java code and MySQL via a database-specific driver.

2.3 Setup — Adding the MySQL Driver

- Download the MySQL Connector/J JAR from dev.mysql.com (e.g. mysql-connector-j-8.4.0.jar).
- For a standalone Java program, add the JAR to your classpath when compiling/running.
- For a web app, drop the JAR into webapps/your-app/WEB-INF/lib/ and restart Tomcat.

2.4 The Five Steps of JDBC

Step	Code
1. Load Driver	Class.forName("com.mysql.cj.jdbc.Driver");
2. Connect	Connection con = DriverManager.getConnection(url, user, pwd);
3. Statement	Statement st = con.createStatement();
4. Execute	ResultSet rs = st.executeQuery("SELECT ...");
5. Close	rs.close(); st.close(); con.close();

2.5 Connection URL Format

Example 2.1 — JDBC URL format

```
jdbc:mysql://HOST:PORT/DATABASE

Examples:
   jdbc:mysql://localhost:3306/bca_portal
   jdbc:mysql://localhost:3306/bca_portal?useSSL=false
   jdbc:mysql://192.168.1.5:3306/college
```

2.6 Worked Example — Read All Students

Example 2.2 — ListStudents.java

```
import java.sql.*;

public class ListStudents {
   public static void main(String[] args) {

      String url  = "jdbc:mysql://localhost:3306/bca_portal";
      String user = "root";
      String pwd  = "";    // default XAMPP password is empty

      try {
         // 1. Load driver
         Class.forName("com.mysql.cj.jdbc.Driver");

         // 2. Connect
         Connection con = DriverManager.getConnection(url, user, pwd);

         // 3. Statement
         Statement st = con.createStatement();

         // 4. Execute
         ResultSet rs = st.executeQuery("SELECT * FROM students");
```

```
            // Loop through the rows
            while (rs.next()) {
                int id      = rs.getInt("id");
                String name = rs.getString("name");
                int marks   = rs.getInt("marks");
                System.out.println(id + "  " + name + "  " + marks);
            }

            // 5. Close
            rs.close();
            st.close();
            con.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

> ***Output:***

```
1  Aman Kumar  85
2  Riya Singh  92
3  Vivek Verma  67
```

2.7 INSERT, UPDATE, DELETE — Use executeUpdate()

Example 2.3 — INSERT / UPDATE / DELETE

```
Statement st = con.createStatement();

// INSERT
int rows = st.executeUpdate(
    "INSERT INTO students (name, course, marks) " +
    "VALUES ('Mohit', 'BCA', 75)");
System.out.println(rows + " row inserted");

// UPDATE
int updated = st.executeUpdate(
    "UPDATE students SET marks = 80 WHERE id = 4");

// DELETE
int deleted = st.executeUpdate(
    "DELETE FROM students WHERE marks < 40");
```

Method	When to use
executeQuery(sql)	Use for SELECT — returns a ResultSet.
executeUpdate(sql)	Use for INSERT/UPDATE/DELETE — returns row count.

execute(sql)	Generic — returns boolean. Rarely used.

3. PreparedStatement — Safe Queries

3.1 The SQL Injection Problem

Building SQL by joining strings with user input is a serious security risk. If a user enters a clever value, they can break out of the query and read or destroy the database. This attack is called SQL INJECTION.

Example 3.1 — SQL injection vulnerability

```
// DANGER — never do this in real code
String name = request.getParameter("name");
String sql = "SELECT * FROM users WHERE name = '" + name + "'";

// If the user types:   ' OR '1'='1
// The SQL becomes:
// SELECT * FROM users WHERE name = '' OR '1'='1'
// → returns ALL users, regardless of name!
```

3.2 PreparedStatement to the Rescue

A PreparedStatement uses placeholders (?) instead of string concatenation. The driver handles escaping safely, so user input can never break out of its slot. PreparedStatement is also FASTER for repeated queries because the database parses the SQL only once.

Example 3.2 — Safe SELECT with PreparedStatement

```
String sql = "SELECT * FROM students WHERE course = ? AND marks >= ?";
PreparedStatement ps = con.prepareStatement(sql);

ps.setString(1, "BCA");      // first ? = course
ps.setInt   (2, 80);         // second ? = marks

ResultSet rs = ps.executeQuery();
while (rs.next()) {
   System.out.println(rs.getString("name"));
}

rs.close();
ps.close();
```

3.3 Inserting with PreparedStatement

Example 3.3 — Safe INSERT

```
String sql = "INSERT INTO students (name, course, marks) VALUES (?, ?, ?)";
PreparedStatement ps = con.prepareStatement(sql);
```

```
ps.setString(1, "Aman");
ps.setString(2, "BCA");
ps.setInt   (3, 85);

int rows = ps.executeUpdate();
System.out.println(rows + " student added");
```

Best practice

ALWAYS use PreparedStatement when the query includes any user input — form fields, URL parameters, headers, anything from outside.

Use plain Statement only for fixed queries with no parameters.

3.4 PreparedStatement Setter Methods

Method	Purpose
setString(i, val)	Bind a String to placeholder i (starts from 1).
setInt(i, val)	Bind an int.
setLong(i, val)	Bind a long.
setDouble(i, val)	Bind a double / float.
setBoolean(i, val)	Bind a boolean.
setDate(i, sqlDate)	Bind a java.sql.Date.
setNull(i, type)	Bind a NULL of the given SQL type.

4. JDBC in a Servlet

4.1 A Realistic Pattern

In a real web app, JDBC code lives inside a Servlet (the controller) or a DAO class (Data Access Object — part of the Model). The result is then passed to a JSP for display, completing the MVC pattern from Week 12.

Example 4.1 — Servlet that fetches all students and forwards to JSP

```
@WebServlet("/students")
public class StudentListServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
            throws ServletException, IOException {
```

```
        List<Student> list = new ArrayList<>();

        String url  = "jdbc:mysql://localhost:3306/bca_portal";
        String user = "root";
        String pwd  = "";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, user, pwd);
            PreparedStatement ps = con.prepareStatement(
                "SELECT id, name, course, marks FROM students");
            ResultSet rs = ps.executeQuery();

            while (rs.next()) {
                Student s = new Student();
                s.setName  (rs.getString("name"));
                s.setCourse(rs.getString("course"));
                s.setMarks (rs.getInt   ("marks"));
                list.add(s);
            }

            rs.close(); ps.close(); con.close();
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Forward to JSP (Week 12 pattern)
        req.setAttribute("students", list);
        req.getRequestDispatcher("list.jsp").forward(req, res);
    }
}
```

Example 4.2 — list.jsp displays the students using JSTL

```
<%-- list.jsp --%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html><body>
  <h1>All Students</h1>
  <table border="1">
    <tr><th>Name</th><th>Course</th><th>Marks</th></tr>
    <c:forEach var="s" items="${students}">
      <tr>
        <td><c:out value="${s.name}"/></td>
        <td><c:out value="${s.course}"/></td>
        <td><c:out value="${s.marks}"/></td>
      </tr>
    </c:forEach>
  </table>
```

```
</body></html>
```

5. XML — Extensible Markup Language

5.1 What is XML?

XML stands for EXTENSIBLE MARKUP LANGUAGE. It is a text-based format for storing and exchanging structured data using tags — similar in syntax to HTML, but where HTML has a fixed set of tags (h1, p, img...), XML lets you invent your own tags. It was the standard format for data exchange before JSON took over.

5.2 Anatomy of an XML Document

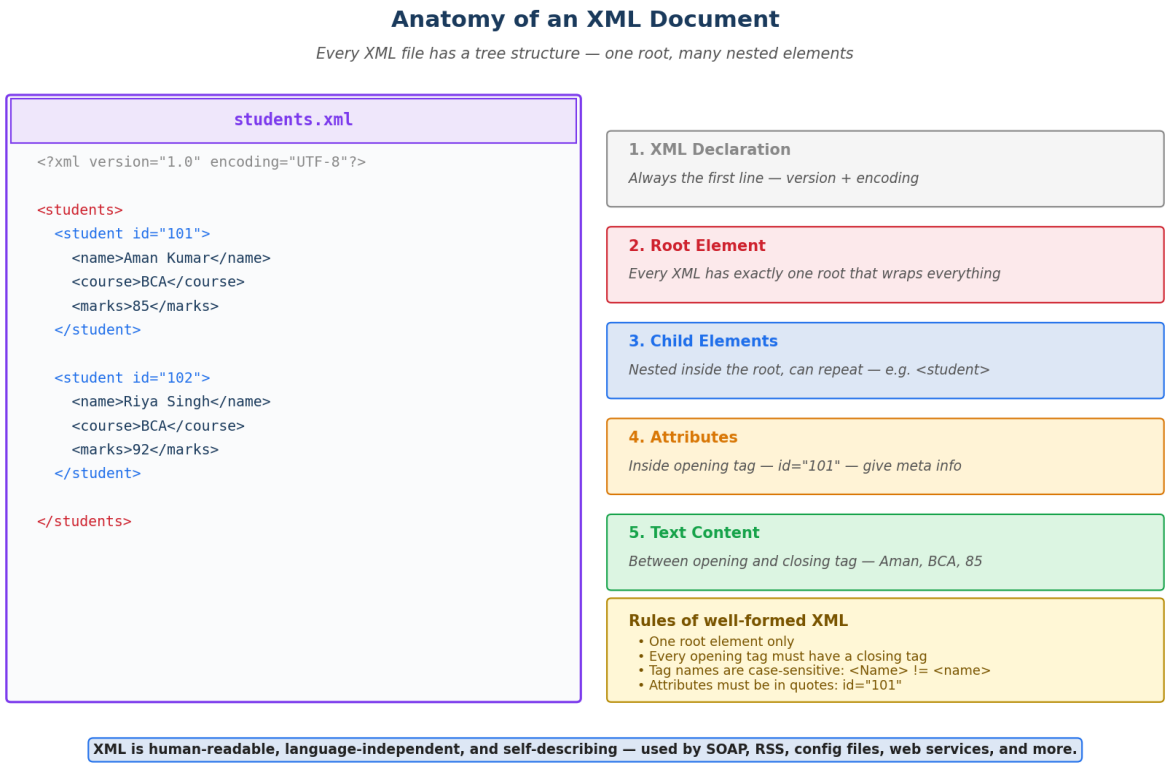


Figure 13.2 — Every XML file has a declaration, a root element, child elements, attributes, and text content.

5.3 Rules of Well-Formed XML

- Exactly ONE root element.
- Every opening tag must have a matching closing tag, e.g. <name>...</name>.
- Tags are CASE-SENSITIVE — <Name> is not the same as <name>.
- Attribute values must be quoted: id="101", not id=101.
- Tags must be properly nested — <a>, never <a>.

5.4 HTML vs XML

Aspect	HTML	XML
Purpose	Display data in a browser.	Carry data between systems.
Tags	Fixed set (h1, p, img, ...).	Make up your own tags.
Case-sensitive?	No.	Yes.
Closing tags?	Often optional ( ).	Always required.
Best for	Web pages.	Configuration files, APIs, data exchange.

5.5 Reading XML in Java

Java's standard JAXP library provides DOM and SAX parsers for XML. Here is a basic DOM parsing example.

Example 5.1 — Reading students.xml with the DOM parser

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.io.File;

public class ReadStudents {
   public static void main(String[] args) throws Exception {
      File f = new File("students.xml");
      DocumentBuilderFactory factory =
         DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = factory.newDocumentBuilder();
      Document doc = builder.parse(f);

      NodeList list = doc.getElementsByTagName("student");
      for (int i = 0; i < list.getLength(); i++) {
         Element st = (Element) list.item(i);
         String id    = st.getAttribute("id");
         String name  = st.getElementsByTagName("name")
                         .item(0).getTextContent();
         String marks = st.getElementsByTagName("marks")
                         .item(0).getTextContent();
         System.out.println(id + "  " + name + "  " + marks);
      }
   }
}
```

> Output:

```
101  Aman Kumar  85
102  Riya Singh  92
```

5.6 Where XML is Used

- Configuration files — Java's web.xml, Maven's pom.xml, Tomcat's server.xml.
- Old-style web services using SOAP.
- RSS feeds for blogs and news sites.
- Document formats — Office documents (.docx) are zipped XML.
- SVG vector images and Android UI layouts.

6. Web Services

6.1 What is a Web Service?

A WEB SERVICE is a server endpoint designed for OTHER programs (not humans) to call over the network. Instead of returning HTML for a browser to display, a web service returns DATA — usually JSON or XML — for an Android app, an iPhone app, another website, or another back-end to consume. This is how the modern web is glued together.

6.2 Why Use Web Services?

- Same data, many clients — one API serves the website, the Android app, the iOS app, and partner systems.
- Language-independent — a Python client can call a Java service and vice versa.
- Loose coupling — the front-end can be redesigned without touching the back-end.
- Microservices architecture — large apps split into many small services that each expose a web API.

6.3 REST vs SOAP

Aspect	SOAP (older)	REST (modern)
Format	XML only.	Usually JSON, sometimes XML.
Protocol	Strict standard with envelopes.	Plain HTTP — uses GET/POST/PUT/DELETE.
Verbosity	Heavy — lots of XML.	Light — short JSON.
Discovery	WSDL contract file.	OpenAPI / Swagger.
Best for	Banking, large legacy enterprises.	Mobile, web, modern microservices.

6.4 REST — Resources and HTTP Verbs

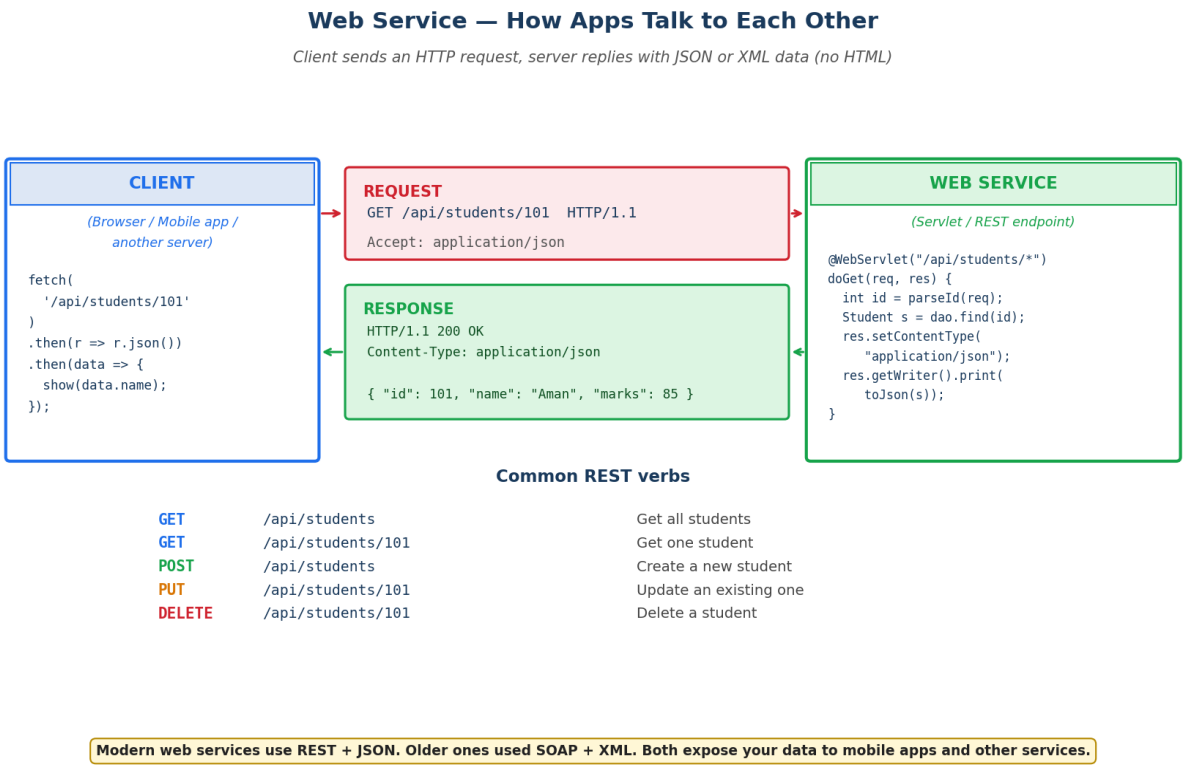


Figure 13.3 — A REST web service call: client sends a GET request, server replies with JSON.

REST stands for REPRESENTATIONAL STATE TRANSFER. The idea is simple: each piece of data is a RESOURCE with a URL like /api/students/101, and standard HTTP verbs decide what to do with it.

Verb + URL	What it does
GET /api/students	Get the list of all students.
GET /api/students/101	Get one student with id 101.
POST /api/students	Create a new student. Data in the body.
PUT /api/students/101	Replace student 101 with new data.
DELETE /api/students/101	Delete student 101.

6.5 Building a REST Endpoint with a Servlet

We can build a basic REST endpoint using the same servlet skills from Week 11, plus JDBC to fetch the data and a small toJson() helper to format it. (For larger projects, frameworks like JAX-RS or Spring Boot make this much easier.)

Example 6.1 — A simple REST endpoint that returns student data as JSON

@WebServlet("/api/students")

```
public class StudentApiServlet extends HttpServlet {

   @Override
   protected void doGet(HttpServletRequest req,
                        HttpServletResponse res)
         throws IOException {

      res.setContentType("application/json");
      PrintWriter out = res.getWriter();

      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/bca_portal",
            "root", "");

         PreparedStatement ps = con.prepareStatement(
            "SELECT id, name, course, marks FROM students");
         ResultSet rs = ps.executeQuery();

         // Build a JSON array manually
         out.print("[");
         boolean first = true;
         while (rs.next()) {
            if (!first) out.print(",");
            first = false;
            out.printf("{\"id\":%d,\"name\":\"%s\",\"marks\":%d}",
               rs.getInt("id"),
               rs.getString("name"),
               rs.getInt("marks"));
         }
         out.print("]");

         rs.close(); ps.close(); con.close();
      } catch (Exception e) {
         res.setStatus(500);
         out.print("{\"error\":\"" + e.getMessage() + "\"}");
      }
   }
}
```

> ***Output:***

```
GET http://localhost:8080/bca-portal/api/students

[
  {"id":1,"name":"Aman Kumar","marks":85},
  {"id":2,"name":"Riya Singh","marks":92},
  {"id":3,"name":"Vivek Verma","marks":67}
```

```
]
```

6.6 Calling the Web Service from JavaScript

Example 6.2 — Same endpoint called from a JavaScript front-end

```
// In a webpage's <script>
fetch('/bca-portal/api/students')
  .then(response => response.json())
  .then(students => {
    students.forEach(s => {
      console.log(s.name + ' scored ' + s.marks);
    });
  })
  .catch(err => console.error(err));
```

7. Worked Example — Complete Database-Backed Result Lookup

This final example brings together everything from the entire course: HTML form (Week 4), CSS (Week 5–7), JavaScript (Week 8–9), Servlet (Week 11), JSP (Week 12), and now JDBC + MySQL (Week 13).

Step 1 — Create the database

Step 1 — SQL setup

```
CREATE DATABASE bca_portal;
USE bca_portal;

CREATE TABLE students (
  id     INT AUTO_INCREMENT PRIMARY KEY,
  name   VARCHAR(50) NOT NULL,
  course VARCHAR(20),
  marks  INT
);

INSERT INTO students (name, course, marks) VALUES
  ('Aman Kumar',  'BCA', 85),
  ('Riya Singh',  'BCA', 92),
  ('Vivek Verma', 'BCA', 67);
```

Step 2 — index.html (front-end form)

Step 2 — index.html

```
<!DOCTYPE html>
<html>
<head><title>Result Lookup</title></head>
<body>
  <h1>BCA Result Lookup</h1>
```

```
    <form action="result" method="GET">
      <label>Roll No:</label>
      <input type="number" name="id" required>
      <button type="submit">Find</button>
    </form>
</body>
</html>
```

Step 3 — ResultServlet.java (controller + JDBC)

Step 3 — ResultServlet.java

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/result")
public class ResultServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req,
                         HttpServletResponse res)
            throws ServletException, IOException {

        int id = Integer.parseInt(req.getParameter("id"));

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/bca_portal",
                "root", "");

            PreparedStatement ps = con.prepareStatement(
                "SELECT name, course, marks FROM students WHERE id = ?");
            ps.setInt(1, id);
            ResultSet rs = ps.executeQuery();

            if (rs.next()) {
                req.setAttribute("name",   rs.getString("name"));
                req.setAttribute("course", rs.getString("course"));
                req.setAttribute("marks",  rs.getInt   ("marks"));
                req.getRequestDispatcher("result.jsp").forward(req, res);
            } else {
                res.getWriter().println("<h2>No student found with id " + id + "</h2>");
            }

            rs.close(); ps.close(); con.close();
```

```
        } catch (Exception e) {
            throw new ServletException(e);
        }
    }
}
```

Step 4 — result.jsp (view)

Step 4 — result.jsp

```
<%@ page contentType="text/html" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
  <title>Result Card</title>
  <style>
    body { font-family: Arial; padding: 30px; }
    .card { border: 2px solid #1f6feb; border-radius: 8px;
            padding: 20px; max-width: 400px; }
  </style>
</head>
<body>
  <h1>BCA Result Card</h1>
  <div class="card">
    <p>Name:   <strong><c:out value="${name}"/></strong></p>
    <p>Course: <c:out value="${course}"/></p>
    <p>Marks:  <c:out value="${marks}"/> / 100</p>
  </div>
</body>
</html>
```

Try this in your lab

1. Start MySQL in XAMPP and run the SQL from Step 1 in phpMyAdmin.
2. Place mysql-connector-j-8.4.0.jar in webapps/bca-portal/WEB-INF/lib/.
3. Compile ResultServlet.java with javac, including the connector and servlet-api jars in the classpath.
4. Place the .class file in WEB-INF/classes/.
5. Save index.html and result.jsp in webapps/bca-portal/.
6. Restart Tomcat and visit http://localhost:8080/bca-portal/index.html.
7. Enter roll 1 → see Aman's result card pulled live from MySQL.

8. Summary of Week 13

- MySQL is a free, open-source relational database. SQL has four main operations: INSERT, SELECT, UPDATE, DELETE (CRUD).

- JDBC is the Java API for connecting to databases. The five steps are: Load Driver → Connect → Statement → Execute → Close.
- Connection URL format: jdbc:mysql://host:port/database.
- executeQuery() returns a ResultSet for SELECT; executeUpdate() returns a row count for INSERT/UPDATE/DELETE.
- Always use PreparedStatement with ? placeholders to prevent SQL injection.
- XML stores structured data in nested tags. Every XML must have one root, balanced tags, and quoted attributes.
- Web services let programs talk to programs. They return data (JSON or XML), not HTML.
- Modern web services use REST + JSON over HTTP. Older ones used SOAP + XML.
- REST resources have URLs like /api/students/101 and use HTTP verbs: GET, POST, PUT, DELETE.
- A simple REST endpoint can be built using a servlet that sets Content-Type to application/json and prints the JSON body.

9. Practice Questions

A. Short Answer (2 marks each)

1. What does SQL stand for? Name the four CRUD operations.

2. Write the SQL to create a 'courses' table with id (auto-increment primary key), name (varchar 50), and credits (int).

3. List the five steps of any JDBC program in order.

4. Differentiate between executeQuery() and executeUpdate().

5. What is SQL injection? How does PreparedStatement prevent it?

6. List any three rules of well-formed XML.

7. Differentiate between REST and SOAP in two points each.

B. Long Answer (5–10 marks each)

1. Explain JDBC architecture with a neat diagram. Describe each of the five steps with one line of code.

2. Compare Statement and PreparedStatement. Write a complete JDBC program that inserts a new student row using PreparedStatement.

3. Write a complete servlet that fetches all rows from a 'products' table and forwards the list to a JSP for display using JSTL.

4. Explain XML — its structure, rules, and use cases. Write a sample XML file containing three student records and a Java DOM-parser snippet to read it.

5. Define a web service. Compare REST and SOAP with examples, and write a complete REST endpoint (servlet) that returns a list of students as JSON.

C. Lab / Hands-On Tasks

1. Set up a 'bca_portal' database in MySQL using phpMyAdmin. Create a 'students' table and insert at least 5 sample rows.

2. Write a Java program that connects to your bca_portal database and prints all student names whose marks are above 80, using PreparedStatement.

3. Build a complete servlet + JSP application that lets a user search for a student by name and displays the matching record(s) in a styled HTML table.

4. Write an XML file describing five courses offered at RVSCET (id, name, duration, fee). Then write a Java program using the DOM parser that prints each course on a separate line.

5. Build a REST endpoint /api/courses that returns the list of courses from a MySQL table as JSON. Test it from the browser, then call it from a JavaScript page using fetch() and display the courses dynamically.

End of the Web Technology course

Congratulations on reaching the end of the 13-week journey! You have travelled from how the internet works (Week 1) all the way to building database-backed REST APIs (Week 13).

Keep building. The best way to remember everything is to build small projects: a personal portfolio, a quiz app, a small admin dashboard, a college notice board. Every line you write makes the concepts more permanent.

All the best for your exams and your career ahead.

References

1. MDN Web Docs. *HTML, CSS, and JavaScript Documentation*. Available at: https://developer.mozilla.org
2. World Wide Web Consortium. *Web Standards Specifications*. Available at: https://www.w3.org
3. PHP Manual. *PHP Documentation*. Available at: https://www.php.net/manual
4. Oracle Corporation. *Java Tutorials*. Available at: https://docs.oracle.com/javase/tutorial
5. Eclipse Foundation. *Jakarta EE Servlet Documentation*. Available at: https://jakarta.ee/specifications/servlet
6. Apache Software Foundation. *Apache Tomcat Documentation*. Available at: https://tomcat.apache.org/tomcat-9.0-doc
7. Oracle Corporation. *MySQL Reference Manual*. Available at: https://dev.mysql.com/doc
8. Bootstrap. *Bootstrap Documentation*. Available at: https://getbootstrap.com/docs

Books

9. Web Technologies. Uttam K. Roy. Oxford University Press.
10. HTML5 Black Book. DT Editorial Services. Dreamtech Press.
11. Internet and World Wide Web: How to Program. Paul Deitel and Harvey Deitel. Pearson Education.
12. Head First HTML and CSS. Elisabeth Robson and Eric Freeman. O'Reilly Media.
13. JavaScript: The Definitive Guide. David Flanagan. O'Reilly Media.
14. PHP and MySQL Web Development. Luke Welling and Laura Thomson. Pearson Education.
15. Servlet & JSP: A Tutorial. Budi Kurniawan.
16. Head First Servlets and JSP. Bryan Basham, Kathy Sierra, and Bert Bates. O'Reilly Media.
17. Java: The Complete Reference. Herbert Schildt. McGraw-Hill Education.
18. Beginning XML. Joe Fawcett, Liam Quin, and Danny Ayers. Wrox Press.
19. RESTful Web Services. Leonard Richardson and Sam Ruby. O'Reilly Media.