CSE326 INTERNET PROGRAMMING

Units 1 & 2 — Summary Notes

From how the Internet works to accessible forms, tables and responsive pages: the whole of Units 1 and 2 in one short revision booklet.

> **How to use these notes.** Read a section, then try its **Check yourself** questions before looking at the answers at the back. The amber **Trap** boxes name the wrong ideas that cost marks.

Contents

UNIT 1 - FOUNDATIONS: THE INTERNET, THE WEB AND HTML5

1. Internet, Web, client-server, HTTP/HTTPS and URLs 2
2. HTML5 document structure, head vs body, meta tags, elements and attributes 5
3. Text formatting, headings and lists 9
4. Hyperlinks, images and link text (SEO) 12
5. SVG graphics 15
6. Multimedia: audio, video and iframes 18

UNIT 2 - SEMANTIC STRUCTURE, ACCESSIBILITY, FORMS AND TABLES

7. Semantic elements and page structure 21
8. Accessibility: POUR, WCAG, ARIA and keyboard access 24
9. Forms, input types and validation 27
10. Tables and data representation 31
11. Responsive content structuring 34

BACK MATTER

Answers to "Check yourself" 37

UNIT 1

Foundations: the Internet, the Web and HTML5

1 Internet, Web, client-server, HTTP/HTTPS and URLs

Textbook chapters 1, 2

The Internet and the World Wide Web are not the same thing

The **Internet** is the wiring of the world: cables, radio links, routers and billions of connected devices, plus the rules (protocols) that move data between them. It is a **network of networks**: your phone joins the home router, the router joins an Internet service provider, and providers join bigger networks across the globe. No single company owns it and no single computer is it.

- Data travels in small **packets**, each labelled with its destination address; routers pass them on like road junctions.
- **TCP/IP** holds it together: IP addresses and moves packets; TCP makes sure they all arrive in order.

The **World Wide Web** (the Web, WWW) is **one service that runs on top of the Internet**: web pages written in HTML, joined by hyperlinks, fetched with HTTP and named by URLs. HTML, HTTP and URLs were invented together at CERN in 1989-91, long after the Internet began. Many other services share the same roads but are **not** the Web, because they use their own protocols and are not made of web pages.

Activity	Uses the Internet?	Part of the Web?	Reason
Reading a news site in a browser	Yes	Yes	HTML pages delivered by HTTP
Sending email from a mail program	Yes	No	Uses mail protocols (SMTP, IMAP)
Playing an online multiplayer game	Yes	No	Uses a game protocol, no web pages
Using a video-streaming app, or a WhatsApp chat or call	Yes	No	Streaming and messaging protocols

Internet = the infrastructure (devices + routers + cables + TCP/IP). Web = one service on it (HTML pages + links + HTTP). Every website is on the Internet, but not everything on the Internet is a website. Email, games and streaming prove the Web is only one of several Internet services.

Trap: Calling the Web the infrastructure and the Internet a service on it is backwards. Saying the two words mean one identical system is false: you can email or game without touching the Web. Saying "the Internet speaks only HTTP and the Web speaks only TCP/IP" swaps the protocols: HTTP belongs to the Web, TCP/IP to the Internet, and HTTP itself travels over the Internet's TCP/IP family of protocols. The Internet is also not one giant computer, not a website holding every page, and not a search engine (a search site like Google is itself one website on the Web). Email and games do not run on HTML either.

Client and server: who asks, who answers

Every web visit is a conversation between two programs. This is the **client-server model**. The **client** asks; the **server** answers.

Part	What it is	Its job	Not its job
Client	The browser (Chrome, Edge, Firefox, Safari) or a phone app	Builds and sends the HTTP request; receives the reply; renders HTML into pixels on screen	Holding the site's database
Server	A program on a computer in a data centre	Stores data, runs logic and responds to requests	Drawing pixels for the user; typing the URL (the user does that)
DNS	Domain Name System, the phone book of the Internet	Turns a domain name into an IP address	Sending the page request or the page
Database	Sits behind the server	Keeps records and answers only the server	Talking to the browser directly

Food-delivery app: your phone (client) shows menus and sends each tap as a request. The company server looks up the restaurant list in its database, works out the delivery time and replies with JSON. The phone never holds the whole database.

What the server sends back is a **response**: a status code plus the content, usually **HTML for a page or JSON for data** (sometimes an image or PDF). It is text, not a picture of the page; the browser interprets and draws it.

> **Trap:** The browser is the client even though it "shows" the page: roles depend on who asks. A server reply never contains your keystrokes, a copy of your browser's cache or history (both live on your own device), or a newly bought domain name (domains come from a registrar, not inside a page). Scrambling what the user types on the keyboard is not the server's job; HTTPS protects data while it travels between the two ends. Moving packets between networks is the routers' job, not the web server's.

HTTP and HTTPS

HTTP = HyperText Transfer Protocol. It is the **protocol used to exchange web requests and responses**: a fixed set of rules for how a request and a reply are worded, so any browser can talk to any server. In HTTP/1.1, the version shown here, the messages are plain text (newer HTTP/2 and HTTP/3 pack the same requests into binary frames):

```
Browser sends:
GET /timetable.html HTTP/1.1
Host: www.citycollege.org

Server replies:
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html><html lang="en"> ...
```

- **GET** asks for a resource; **POST** sends data (such as a form) in the body.
- Status codes by first digit: 2xx success (200 OK), 3xx redirect, 4xx client error (404 Not Found), 5xx server error.

Feature	HTTP	HTTPS
Full form	HyperText Transfer Protocol	HTTP Secure
Data on the way	Readable text; anyone on the path (cafe Wi-Fi, ISP) can read or change it	Encrypted with TLS (Transport Layer Security); a certificate proves the server is genuine
Default port	80	443
Address bar	"Not secure" warning	Padlock or site-info icon (no warning)

> HTTPS = the same HTTP conversation in a sealed envelope: HTTP plus TLS encryption of the data in transit. Only the scheme and the default port (443) change; the path, the HTML or JSON content and the server code stay the same.

> **Trap:** HTTP is a protocol. It is not the physical cables (that is infrastructure), not a language for querying databases (that is SQL), and not a markup language for building pages (that is HTML). HTTPS does not compress or shrink files, does not fit layouts to phone screens (that is responsive CSS), and does not add server-side scripting (PHP or Node.js code runs with or without it).

The request-response cycle, step by step

1. **Click**: the user clicks a link or types a URL.
2. (DNS lookup: the browser turns the domain name into an IP address.)
3. **Request**: the browser (client) sends an HTTP request to that server.
4. **Server processes**: it reads a file, or runs code and queries its database.
5. **Response**: it sends back a status code and the content (HTML or JSON).
6. **Render**: the browser draws the page, requesting each CSS file, image and script the HTML mentions (unless already cached).

> Click, then request, then server processes, then response, then render. This order never changes.

> **Trap:** Reject any order where the request comes before the user acts, the response comes before the request, or the server processes before anyone clicks.

Reading a URL part by part

A **URL** (Uniform Resource Locator) is the address the browser requests. Read it like a postal address: how to deliver, which building, which room, plus a note.

```
https://lib.citycollege.org/books/find.html?title=html&year=2026
|       |                   |               |
|       |                   |               +-- query string (extra data)
|       |                   +-- path (which file)
|       +-- domain (which server)
+-- protocol (how to fetch)
```

Part	Example	Job	How to spot it
Protocol (scheme)	`https`	How to fetch: http or https (also mailto, tel)	Before `://`
Domain (host)	`lib.citycollege.org`	Which server; DNS turns it into an IP address	After `://`, up to the first `/`
Path	`/books/find.html`	Which resource on that server, like a folder path	Starts with `/`, runs up to `?`
Query string	`?title=html&year=2026`	Extra name=value parameters for the server, joined by `&`	Starts with `?`, runs up to `#` or the end
Port (optional)	`:8080` (not in this URL)	Which "door" on the server; defaults are 80 (HTTP) and 443 (HTTPS)	Right after the domain
Fragment (optional)	`#reviews` (not in this URL)	Jumps to the element with that id; never sent to the server	Starts with `#`

Worked example: `https://results.techuni.ac.in/sem3`
Protocol = `https`; domain = `results.techuni.ac.in` (the whole host name); path = `/sem3`; no query string.
Add `?roll=45` to the end and that new piece is the query string; the path stays `/sem3`.

Trap: The domain is the full host name, not just its last label. In `results.techuni.ac.in` the ending `in` is only the top-level domain (India's country code). `https://` with its colon and slashes marks the protocol; it is never the path. A path always begins with `/` and a query string always begins with `?`.

An **absolute URL** has the protocol and domain and means the same thing anywhere; a **relative URL** such as `find.html` is completed by the browser from the current page's address.

CHECK YOURSELF

1.1 A student's laptop has no browser open. In one hour the student sends mail from a desktop mail program (SMTP/IMAP), plays an online multiplayer match and joins a WhatsApp call. Which conclusion is correct?

a. The student used the Web, because any device with an IP address is on the Web
b. The student used the Internet but not the Web, since none of it fetched HTML pages over HTTP
c. The student used neither, because nothing can reach the Internet without a browser
d. The student used the Web, because SMTP and IMAP are the Web's own page protocols

1.2 A student opens a recipe app and taps 'Paneer tikka'. Which task is done by the recipe company's web server?

a. Keeping the phone's own browsing history up to date
b. Turning the app's domain name into an IP address
c. Looking up the recipe in its database and returning it as JSON
d. Routing the data packets through the ISP's routers

1.3 A college moves its library site from http:// to https://. What actually changes for a visitor?

a. The URL path changes, for example /home becomes /secure/home
b. The server must start sending JSON instead of HTML pages
c. The page's HTML must be rewritten in a new markup language
d. The browser uses port 443 and data in transit is TLS-encrypted

1.4 Which of these is an example of HTTP itself at work?

a. The text line GET /notice.html HTTP/1.1 sent from a browser to a server
b. The IP address 203.0.113.25 that a DNS lookup returns
c. The CSS rule p { color: navy; } that styles a paragraph
d. The Wi-Fi radio signal between a phone and its home router

1.5 On an e-book site: (P) the server finds Chapter 5 and builds the HTML; (Q) the browser draws the chapter; (R) the browser sends a request for /chapter-5.html; (S) the server returns 200 OK with the HTML; (T) the reader taps 'Next chapter'. Which order is correct?

a. R, T, P, S, Q
b. T, S, R, P, Q
c. T, R, P, S, Q
d. P, T, R, S, Q

1.6 In http://events.citycollege.org/fest/2026/schedule.html?day=2&hall=B, which label is correct?

a. Query string: /fest/2026
b. Path: /fest/2026/schedule.html
c. Protocol: events
d. Domain: day=2

1.7 For https://admissions.greenvalley.ac.in/apply?course=btech, a friend says the domain is just 'in' and the protocol is '/apply'. What is actually true?

a. The domain is admissions.greenvalley.ac.in and the protocol is https
b. The domain is in and the protocol is https
c. The domain is /apply and the protocol is admissions
d. The domain is ?course=btech and the protocol is ac

2 HTML5 document structure, head vs body, meta tags, elements and attributes

Textbook chapters 3, 4, 5, 11, 12

The skeleton, the doctype and the root element

Every HTML5 page starts with the same skeleton. Learn it once and reuse it in every file.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hostel Mess Menu</title>
</head>
<body><h1>This Week's Menu</h1></body>
</html>
```

- **Line 1,** `<!DOCTYPE html>`, tells the browser to use the modern HTML5 parsing rules (standards mode). It is a *declaration*, not an element, and has no closing tag. Case does not matter: `<!doctype html>` works too. Leave it out and the browser drops into **quirks mode**, copying 1990s layout rules, so spacing and box sizes go subtly wrong.
- `<html>` **is the root element**: the single container that holds everything else. It has exactly two children, `<head>` first and then `<body>`. Head and body are siblings inside html, so neither is the root. The doctype sits above html but is not an element at all, so it cannot be the root either.
- `lang="en"` on html declares the language the content is written in (`hi` = Hindi, `pa` = Punjabi). Screen readers use it to choose the right pronunciation; browsers use it to offer translation and to pick spell-check and hyphenation rules. It does not translate the page, restrict fonts or stop anyone typing other characters.

> **Trap:** `<!DOCTYPE HTML5>`, `<html5>` and `<meta doctype="html5">` do not exist. The HTML5 doctype carries no version number, just the word html. A doctype is never a meta tag and never an element.

The browser reads the page as a tree

The browser does not keep your file as one long string, a flat list of separate tags, or a grid of rows and columns. It parses the markup into a nested tree of elements, called the **DOM** (Document Object Model). Every element is a node with one parent and possibly children; CSS and JavaScript work by walking this tree, and DevTools (F12, Elements panel) shows it.

```
html
 +-- head
 |    +-- meta
 |    +-- title
 +-- body
      +-- h1
```

Because it is a tree, nesting must be clean: **last opened, first closed**. `<p><em>hi</em></p>` is right; `<p><em>hi</p></em>` crosses over and forces the browser to guess.

Head or body: one question decides

The **head** holds *metadata*, information about the page; nothing in it is drawn on the page. The **body** holds everything the reader sees or uses. Ask one question: is it drawn on screen? Yes means body; information about the page means head.

Belongs in head (invisible)	Belongs in body (visible)
`<title>Exam Cell</title>`	`<h1>` to `<h6>` headings
`<meta charset="UTF-8">`	`<p>` paragraphs
`<meta name="viewport" ...>`	`<img>` photos, `<video>`
`<meta name="description" ...>`	`<table>`, `<form>`, lists
`<link rel="stylesheet" href="site.css">`, `<link rel="icon">`	`<header>`, `<nav>`, `<main>`, `<footer>`
`<style>` blocks	`<a>` links, `<button>`

`<script>` may go in either place: in the head with `defer` (so it waits until the page is parsed), or just before `</body>`. It is never drawn, so it is the one common exception to the rule.

> **Trap:** a heading, paragraph or image placed in the head is a mistake, however small. The browser quietly ends the head and pushes it into the body, and the page is invalid. Title, meta and a stylesheet link are all correct in the head. And `<head>` is not `<header>`: header is a visible banner inside the body.

The three meta tags to know

`<meta>` is a void element (no closing tag) that supplies one piece of metadata. Each of these three has exactly one job:

Tag	Its one job	Where you notice it
`<meta charset="UTF-8">`	Declares the character encoding: how the browser turns the file's bytes into characters	Rupee signs, accents and Hindi text display correctly instead of junk such as `â‚¹`
`<meta name="viewport" content="width=device-width, initial-scale=1.0">`	Lays the page out at the phone's real width, 100% zoom	Without it a phone shows a tiny, shrunken desktop page
`<meta name="description" content="...">`	A short summary of the page	Typically shown as the summary text under the clickable headline in search results

- charset is **not** about fonts (fonts are CSS `font-family`), not about phones (viewport) and not the search summary (description). Put it first in the head.
- The description is **not** shown in the browser tab (that is the title) or as a heading on the page (that is h1). It is not invisible either: people read it on the search results page, where a good one wins the click. Engines sometimes show other text from the page instead.

The title element versus the h1 heading

	`<title>`	`<h1>`
Lives in	head	body
Kind	Metadata	Visible page content

Seen as	Browser tab text, bookmark name, clickable headline in search results	The large main heading on the page
How many	One per document, never more	Normally one main heading

```
<head><title>Library Timings | Central Library</title></head>
<body><h1>Library Timings</h1></body>
```

The tab reads "Library Timings | Central Library"; the page shows the big heading "Library Timings". The title is never drawn on the page, so the two never render alike.

Trap: swapping the roles (calling h1 metadata and title the visible content) gets it backwards, and a document may not have several title elements.

Elements versus attributes

Elements express meaning; attributes configure them. The element says what the thing is (an image, a link, a heading). Attributes add the details (which file, where to go, which language). The two are not interchangeable, and the reversed sentence (attributes give meaning, elements configure) is backwards.

```
<abbr title="Cascading Style Sheets">CSS</abbr>
```

Part	Here	Rule
Opening tag	`<abbr title="...">`	Starts the element; the only place attributes may go
Attribute	`title="Cascading Style Sheets"`	A `name="value"` pair, value usually in quotes (a boolean attribute such as `required` needs no value)
Content	`CSS`	What sits between the tags
Closing tag	`</abbr>`	Ends the element; never carries attributes

Trap: a pair such as `target="_blank"` or `alt="..."` written inside an opening tag is an attribute. It is not the content (that sits between the tags), not the closing tag, and not a CSS declaration. A CSS declaration looks like `color: red;` (property, colon, value) and belongs in a stylesheet.

Global attributes can be applied to almost any element: `id` (a unique name for one element), `class` (a reusable group name), `style`, `title` (tooltip), `lang`, `hidden` and `data-*`. Most other attributes belong to one family of elements:

Attribute	Used on (main cases)	Means
`href`	`<a>`, `<link>`, `<area>`, `<base>`	A destination or address to go to
`src`	`<img>`, `<video>`, `<audio>`, `<source>`, `<script>`, `<iframe>`	A file to load and embed
`action`	`<form>`	Where to submit the data
`alt`	`<img>`, `<area>`, `<input type="image">`	Text alternative for the picture

So for an attribute that works on almost any element, choose a global one such as id; href, src and action are tied to links, embedded files and forms.

Block-level and inline elements

Block-level elements start on a new line and stretch across the full width, stacking like bricks. **Inline** elements flow inside a line of text and take only the width of their content, like one bold word in a sentence. CSS can change either, but these are the defaults.

Block-level by default	Inline by default
`<div>`, `<p>`, `<h1>` to `<h6>`, `<ul>`, `<li>`, `<section>`, `<table>`, `<form>`	`<span>`, `<a>`, `<strong>`, `<em>`, `<b>`, `<code>`, `<img>`, `<label>`

`<div>` and `<span>` carry no meaning: div is the generic block container and span the generic inline one, for a few words inside a line.

```
<p>Last date: <span class="due">31 March</span> for fee payment.</p>
```

The span lets CSS colour just the date without breaking the line. Usually inline goes inside block. The main exception is `<a>`, which in HTML5 may wrap block content such as a whole card. A `<p>`, however, can never contain a `<div>`, so the browser closes the paragraph early.

Repairing a broken fragment

Broken:

```
<h2>Canteen Menu<h3>
<img src="thali.jpg">
<a>Order online</a>
```

Fixed:

```
<h2>Canteen Menu</h2>
<img src="thali.jpg" alt="Veg thali with rice, dal and roti">
<a href="order.html">Order online</a>
```

Three faults, three targeted fixes: (1) the heading was never closed and a second heading was opened instead, so write the closing tag that matches the opening name; (2) the image has no alt, so screen readers and search engines cannot tell what it shows; (3) the link has no href, so it goes nowhere.

Trap: half-fixes look finished but are not. A closing tag must repeat the opening name exactly: `<h2>` closes with `</h2>`, never `</h3>`. An empty `href=""` only points back to the current page. Deleting the link text, the image's `src` or the heading loses content, and unrelated markup (a div for the heading, a title in the body, a bordered table) fixes nothing. Fix the real fault and keep the content.

CHECK YOURSELF

2.1 A page renders with odd spacing in every browser. View Source shows that its very first line is <html lang="en">. Which line should be added above it so the browser uses modern standards-mode rules?

a. <html version="5">
b. <meta standard="html5">
c. <!doctype html>
d. <head type="html5">

2.2 A teammate's <head> section contains the four items below. Which one is in the wrong place and belongs in the <body>?

a. <meta name="author" content="Web Club">
b. <table> listing the campus bus timings
c. <link rel="icon" href="logo.png">
d. <style>p { margin: 0; }</style>

2.3 On an online shop, the rupee sign and Hindi product names show up as junk such as â,[1]. Which head line is meant to prevent exactly this?

a. <meta charset="UTF-8">
b. <meta name="viewport" content="width=device-width, initial-scale=1.0">
c. <link rel="stylesheet" href="hindi-fonts.css">
d. <meta name="keywords" content="rupee, hindi">

2.4 A search for 'central library hours' lists a college site. Under its clickable headline sits the line 'Open 8 am to 10 pm, borrow up to five books.' Which part of the page's HTML most likely supplied that line?

a. The <h1> inside the body
b. The <title> element in the head
c. The alt text of the site logo
d. The content of <meta name="description">

2.5 Look at <img src="hostel.jpg" alt="Hostel block at night" width="300">. Which statement is correct?

a. alt="Hostel block at night" is the element's content
b. width="300" is a CSS declaration inside the tag
c. src, alt and width are attributes of the img element
d. The three pairs belong on a closing </img> tag

2.6 Which of these elements, with no CSS applied, starts on a new line and stretches across the full width of its container?

a.
b. <section>
c. <a>
d. <code>

2.7 A lab-safety page contains <h3>Safety Rules<h3> followed by <img src="goggles.png">. Which repaired version fixes both faults and keeps all the content?

a. <h3>Safety Rules</h4> <img src="goggles.png" alt="Pair of clear safety goggles">
b. <h3>Safety Rules</h3> <img alt="Pair of clear safety goggles">
c. <h3 alt="Pair of clear safety goggles">Safety Rules</h3> <img src="goggles.png">
d. <h3>Safety Rules</h3> <img src="goggles.png" alt="Pair of clear safety goggles">

3 Text formatting, headings and lists

Textbook chapters 6, 7, 12

Headings build the outline, not the font size

HTML has six heading levels, `<h1>` (highest) to `<h6>` (lowest). They are the page's contents list: h1 the title, h2 the chapters, h3 the sections inside a chapter. Bigger text is only a side effect. Screen-reader users jump from heading to heading, and search engines read headings to learn what the page is about.

- Exactly one h1 per page, naming the page's main topic. Not one per paragraph, not "as many as look nice", and never none: headings are structure, not decoration. A second h1 is not a syntax error, but it blurs which topic is the main one.
- Do not skip levels. After h1 comes h2, then h3. Jumping from h1 straight to h4 leaves holes in the outline that screen-reader users and search engines both rely on.
- Never pick a level for its size. Want a smaller subheading under an h2? Use h3 and shrink it with CSS.

```
<h1>City Public Library</h1>
<h2>Timings</h2>
<h3>Weekend hours</h3>
<h2>Membership</h2>
```

Trap: "Skipping to h4 is wrong because browsers will not display it." False. Browsers render all six levels, `<h4>` is fully valid in HTML5, and headings do not have to sit inside a single `<section>`. The real fault is the broken outline. Faking a heading with `<p><b>` or `<div class="h2">` throws the outline away completely.

Meaning versus looks: strong/b and em/i

Logical (semantic) elements say what text means; physical (presentational) elements only say how it looks. Both families exist in HTML5 and look identical on screen, so choose by meaning: a screen reader can act on meaning, never on "make this bolder".

Element	What it means	Typical use
`<strong>`	Strong importance, seriousness or urgency	A safety warning, a deadline
`<b>`	Bold ink to draw the eye, with no extra importance	A product name or keyword
`<em>`	Stressed emphasis that changes how the sentence is spoken and what it means	`I asked for <em>tea</em>, not coffee.`
`<i>`	A different kind of text, with no stress	A species name, a technical term, a foreign phrase
`<u>`	A non-textual annotation, such as marking a misspelling	Rarely; underlined text looks like a link
`<small>`	Fine print and side comments	"Taxes extra", a copyright line

```
<p><strong>Warning:</strong> unplug the iron after use.</p>
<p>Our new model is the <b>Nova X</b>.</p>
<p>The tiger is <i>Panthera tigris</i>.</p>
```

strong = "say it with urgency"; b = "heavier ink". em = "stress this word"; i = "a different voice". Both strong and b are current HTML5 elements, and they do not mean the same thing.

Trap: Swapping the pair ("`<b>` means important, `<strong>` is just bold") or believing `<strong>` was removed from HTML5. For stressed emphasis, `<i>` is wrong (no stress), `<u>` is wrong (not emphasis at all) and `<small>` is wrong (fine print). Use `<i>` for "a different kind of text", never just to get italics.

The rest of the formatting kit

Element	Job
`<p>` / ` ` / `<hr>`	A paragraph / a line break inside one block (address lines, poem lines) / a thematic break where the topic changes
`<pre>` / `<mark>`	Keeps spaces and line breaks exactly as typed / highlights relevant text, like a marker pen
`<del>` / `<ins>`	Removed text (struck through) / inserted text (underlined)
`<sub>` / `<sup>`	Lowered text, the 2 in H2O / raised text, the 2 in x squared
`<abbr title="...">`	An abbreviation; the title attribute holds the full form
`<q>` / `<blockquote>`	A short inline quote (the browser adds the quote marks) / a long quote as its own block
`<cite>`	The title of a book, film or other work
`<code>` / `<kbd>` / `<samp>`	Inline code / keys the user presses / output from a program

Trap: Using `<br><br>` to space two ideas apart. Two ideas are two `<p>` elements, and spacing is CSS's job. Look-only tags such as `<font>`, `<center>`, `<big>` and `<strike>` are obsolete: use CSS, or `<del>` for removed text.

Three kinds of list: ask "does the order matter?"

The content	List type	Markup
Items whose order carries no meaning: skills, things to pack, menu links	Unordered (bullets)	`<ul>` + `<li>`
Sequence or rank matters: steps to follow, a recipe in cooking order, a top-five ranking	Ordered (numbers)	`<ol>` + `<li>`
A term paired with its description: glossary, FAQ, "Name: value" details	Description list	`<dl>` holding `<dt>` (term) and `<dd>` (description)

```
<dl>
  <dt>Browser</dt>
  <dd>A program that fetches and shows web pages.</dd>
  <dt>Server</dt>
  <dd>A computer that stores pages and sends them on request.</dd>
</dl>
```

- The browser numbers an ol by itself; never type "1." yourself.
- "The items are related" is no reason for ul: steps are related too, but their order is the information.
- Numbers are no reason for a table. Tables are for grids of data in rows and columns.
- A description list never uses li. Its children are dt and dd pairs.

Trap: Choosing `<dl>` "because each step has a description". Every step has words; that does not turn a sequence into term-and-definition pairs. A leaderboard of the top five scorers is an `<ol>` too: the position is the information.

Numbering an ordered list: type, start, reversed

Attribute on ol	Markers produced
`type="1"` (the default)	1, 2, 3

`type="a"` / `type="A"`	a, b, c / A, B, C
`type="i"`	i, ii, iii (lowercase Roman)
`type="I"`	I, II, III (uppercase Roman)
`start="4"`	Counting begins at 4: 4, 5, 6 (with type="I": IV, V, VI)
`reversed`	Numbers count down (3, 2, 1); the items stay in the order written

Memory rule: the case you type is the case you get. Capital I gives capital Roman numerals; small i gives small ones. `<li value="10">` sets one item's number, and later items continue from it. Bullet shapes on a ul are changed with CSS (list-style-type), not with type.

```
<ol type="I">
  <li>Introduction</li>
  <li>Survey</li>
  <li>Findings</li>
</ol>  <!-- shows I. II. III. -->
```

Nesting a list inside a list

Only li elements may sit directly inside a ul or ol. Loose text, a p, or a whole second list placed straight inside the list is invalid. An li, however, can hold anything, including a complete sub-list. So a sub-list goes inside the li it belongs to: after that item's text, before its closing tag.

```
<ul>
  <li>Fruits
    <ul>
      <li>Mango</li>
      <li>Guava</li>
    </ul>
  </li>
  <li>Vegetables</li>
</ul>
```

Broken pattern	Why it is invalid
`<ul><li>Fruits</li><ul><li>Mango</li></ul></ul>`	The inner ul is a direct child of the outer ul, and the Fruits item closed before its sub-list
`<ul><ul><li>Mango</li></ul></ul>`	The same fault, with no parent item at all
`<ul>Fruits<li>Mango</li></ul>`	Loose text ("Fruits") directly inside a ul

> Egg-tray rule: a list holds only li "eggs". A paragraph, an image or a sub-list goes inside an egg, never loose in the tray.

CHECK YOURSELF

3.1 A recipe page has three phrases: (1) the serious warning "Hot oil can cause severe burns", (2) the Italian cooking term "al dente", and (3) the sentence "Add the salt after the water boils", where the cook stresses the word after. Which elements suit phrases 1, 2 and 3, in that order?

a. <b>, , <i>
b. , <i>,
c. , , <b>
d. <mark>, <u>, <small>

3.2 A robotics club page has its headings in this order: h1 "Robotics Club", h2 "Projects", h4 "Line-follower bot", h2 "Join us". Which single change gives the page a correct outline?

a. Change the h4 to an h3
b. Turn "Join us" into a second h1 so it stands out
c. Change the h1 to an h2 so every heading starts at the same level
d. Leave it as it is, because any level is fine if the text size looks right

3.3 A canteen website shows: (1) today's snacks, in no particular order; (2) how to order: take a token, pay at the counter, collect the food; (3) each dish name followed by a short description of what is in it. Which list elements match 1, 2 and 3?

a. , <ul>, <dl>
b. , <dl>,
c. <dl>, <ol>, <table>
d. , <ol>, <dl>

3.4 A course outline numbers its units with capital Roman numerals and shows the topics HTML and CSS as a sub-list under the first unit. Which markup is valid and produces those markers?

a. <ol type="I">
Unit one
HTML
CSS
b. <ol type="i">
Unit one
HTML
CSS
c. <ol type="I">
Unit one
HTML
CSS
d. <ol type="1">
Unit one
HTML
CSS

4 Hyperlinks, images and link text (SEO)

Textbook chapters 8, 11, 12

The anchor element and where href can point

A link is an `<a>` (anchor) element. Its href attribute (hypertext reference) holds the destination; the words between the tags are the clickable link text. No href, no link. Do not confuse it with `<link>`, which sits in the head and attaches resources such as a stylesheet or the tab icon; it never makes anything clickable.

href value	Where it goes
`https://www.w3.org/`	Absolute URL: another site (protocol and domain included)
`fees.html`	A file in the same folder as the current page
`../index.html`	Up one folder to the parent, then that file
`/index.html`	Start from the root folder of the site
`#timetable`	Fragment: the element with id="timetable" on this same page
`rules.html#late-fee`	Another page, scrolled to its id="late-fee" element

```
<a href="#timetable">Jump to the timetable</a>
<!-- ...a long page... -->
<section id="timetable">
  <h2>Timetable</h2>
</section>
```

Trap: Only the hash sign (#) means "on this page". Without it, `href="timetable"` asks for a file named timetable in the current folder, and `href="/timetable"` asks for one at the site root. A leading dot, as in `.timetable`, is CSS class-selector style; in an href it is just an odd file name, not a jump. A fragment link needs a matching id attribute; if none exists, the click silently fails and the page does not scroll.

Email and phone links: mailto: and tel:

Two special schemes hand the click to another app on the visitor's device:

- `mailto:` opens the visitor's email program with a new message to that address.
- `tel:` lets a phone user dial the number with one tap. Include the country code (+91 for India).
- `sms:` also exists, but it opens a text message rather than a call.

```
<a href="mailto:office@school.example">Email the office</a>
<a href="tel:+911234567890">Call the office</a>
```

Trap: The schemes are spelled exactly mailto: and tel:. Look-alikes such as email:, mail:, send:, call:, phone: and dial: are not recognised, so the browser has no app to hand them to and the click does nothing useful.

Opening a new tab safely: target and rel

- `target="_blank"` opens the link in a new tab (the default, `_self`, reuses the tab). Use it sparingly: the new tab has no Back history, so the visitor cannot press Back to return to your page.
- `rel` states the relationship between this page and the linked one. Rule: whenever a new tab shows an outside or untrusted page, pair it with rel="noopener noreferrer".
- noopener: the new page cannot reach back through `window.opener` and control your tab (for example, quietly swapping it for a fake login page). noreferrer: the new site is not told which page the visitor came from.

```
<a href="https://shop.example.net" target="_blank"
   rel="noopener noreferrer">Visit the partner shop (opens a new tab)</a>
```

The HTML standard and today's browsers already treat target="_blank" as if noopener were written. Adding rel="noopener noreferrer" yourself is still the recommended habit: it protects visitors on older browsers, and noreferrer also hides which page the visitor came from. It is a safety measure; it does not make the link load faster.

Trap: rel="stylesheet" belongs on `<link>` for attaching CSS. The download attribute saves a file instead of opening it, which gives no protection. There is no referrer="none" attribute; the real one is called referrerpolicy.

Link text that names the destination

Screen-reader users often call up a list of every link on the page; "click here, more, here, »" tells them nothing. Search engines also read link text as a label for the page it points to. So link text should say where the link goes or what it gets you, and make sense on its own.

Weak	Strong
`<a href="fees.html">Click here</a>`	`<a href="fees.html">See the hostel fee structure</a>`
"Read more", "Link", "»"	"Download the 2026 academic calendar (PDF)"

Descriptive link text and descriptive alt text help SEO and accessibility at the same time; the two checklists largely overlap. A title tooltip cannot rescue vague link text: touch-screen and keyboard users never see it.

The image element: src and alt

`<img>` is a void element (no closing tag). Treat two attributes as mandatory: src, the resource location (a relative path or a full URL), and alt, a meaningful text alternative. In short: src says where the file is, alt says what the image means.

```
<img src="images/lab.jpg"
     alt="Students soldering circuits in the electronics lab"
     width="600" height="400" loading="lazy">
```

alt does three jobs: a screen reader speaks it, the browser shows it in place of a picture that fails to load, and search engines use it to index the image. Other jobs belong to other attributes and elements:

Need	Use
A caption under a picture	`<figure>` with `<figcaption>`
Reserve space so the page does not jump while loading	width and height
Fetch only when the user scrolls near	loading="lazy"
A hover tooltip / a CSS hook	title / class
The file size	No attribute states it

To make an image a link, put the img inside an a. Its alt then becomes the link text, so it should name the destination.

Trap: no alt, and the file breaks. Without alt the image has no text alternative. A sighted visitor sees a broken-image icon or just a gap, depending on the browser; a screen-reader user gets nothing useful, often just the file name read aloud. A title attribute, if one happens to exist, is at best a weak fallback, not a guaranteed replacement, and by default the browser does not write a description for you.

Writing alt text for each kind of image

Imagine reading the page to a friend over the phone: whatever you would say in place of the picture is its alt text.

Kind of image	alt should be	Example
Informative (photo, chart)	The useful information it shows	`alt="Bar chart: admissions rose from 800 to 1,200"`
Purely decorative (divider, flourish)	Empty but present, so screen readers skip it silently	`<img src="wave.png" alt="">`
Functional (an icon used as a button or link)	The action it performs, not its look	`<button><img src="bin.svg" alt="Delete"></button>`
A logo carrying the organisation's name	The organisation's name	`alt="Green Valley College"`

So a printer icon on a print button gets alt="Print": that is what pressing it does.

> **Trap:** For decoration, leaving alt out is not the same as alt="": a missing alt can make a screen reader read the file name. Filler such as alt="image" or alt="decorative picture" is noise that gets read aloud. For a functional icon the opposite holds: alt="" would leave the button with no name, and describing its colours, shape or file name does not say what the button does.

Responsive images: srcset, sizes and picture

A phone on mobile data should not download the huge photo a desktop monitor needs.

- `srcset` lists several image files, each with its real width (400w means 400 pixels wide). The browser picks the one that suits the screen's size and sharpness.
- `sizes` tells the browser how wide the image will be displayed, which it needs to make that choice.
- `src` remains the fallback for browsers that ignore srcset.

```
<img src="hall-800.jpg"
     srcset="hall-400.jpg 400w, hall-800.jpg 800w, hall-1600.jpg 1600w"
     sizes="(max-width: 600px) 100vw, 600px"
     alt="Convocation hall on graduation day">
```

`<picture>` goes further: several `<source>` elements, each with a condition (a media query or a file type), plus an `<img>` inside as the required fallback. The browser uses the first source that matches.

> **Trap:** srcset is not the alt text (that is alt), not a CSS class (that is class), and it does not push the image to load last (delaying off-screen images is loading="lazy"). It is a list of candidate files.

CHECK YOURSELF

4.1 A long FAQ page has <section id="refunds"> near the bottom. Which link at the top of the same page scrolls straight to that section?

a. Refunds
b. Refunds
c. Refunds
d. Refunds

4.2 A student's portfolio links to an outside online shop using target="_blank" rel="noopener noreferrer". What does the rel value protect against?

a. The shop page loading more slowly than the portfolio
b. The shop tab taking control of the portfolio tab
c. Search engines listing the shop page in results
d. The browser saving the shop page as a download

4.3 A hostel contact card needs one link that starts a phone call to the warden and one that opens a new email to the hostel office. Which pair of href values is correct?

a. href="telephone:+911800123456" and href="mailto:office@hostel.example"
b. href="tel:+911800123456" and href="inbox:office@hostel.example"
c. href="sms:+911800123456" and href="mailto:office@hostel.example"
d. href="tel:+911800123456" and href="mailto:office@hostel.example"

4.4 A sports page has three images: a photo of the captain lifting a trophy, a decorative wavy line between sections, and a magnifying-glass icon that is the only content of the search button. Which set of alt values is best?

a. Photo: alt="Captain lifting the inter-college football trophy"; line: alt=""; icon: alt="Search"

b. Photo: alt="captain.jpg"; line: alt="wavy line"; icon: alt="magnifying glass"

c. Photo: alt="Captain lifting the inter-college football trophy"; line: no alt attribute at all; icon: alt="Magnifying glass icon"

d. Photo: alt="picture"; line: alt=""; icon: alt=""

4.5 A news site wants phones to download a 500-pixel-wide version of a match photo and large monitors a 1500-pixel version, both from one <img>. Which attribute holds the list of those candidate files?

a. sizes

b. srcset

c. loading

d. media

5 SVG graphics

Textbook chapter 9

Raster versus vector: why a logo should be SVG

A JPG or PNG is a **raster** image: a grid of coloured pixels. Enlarge it and the browser has to invent pixels that were never stored, so edges turn blurry or blocky. An **SVG** (Scalable Vector Graphics) file stores drawing instructions instead, such as "circle, centre here, this radius, orange". The browser redraws those instructions at whatever size is needed, so the picture remains crisp however far you zoom and on any size of screen.

Photo versus knitting pattern: enlarge a photo and you see grain; knit the pattern in thicker wool and you get the same design, bigger and just as neat. So logos, icons and diagrams are SVG; photographs stay JPG.

Claim about SVG	Verdict
Looks equally crisp on a phone, a laptop and a projector	True: the real reason to choose it for a logo
Guaranteed to weigh less than the JPG version	False: simple shapes are tiny, but a very detailed drawing can outweigh a photo
Can show colours that JPG cannot	False: both can show millions of colours; colour range is not the reason
Protected from being saved or copied	False: it is plain text that anyone can save

Use it as a file (`<img src="badge.svg" alt="Robotics Club">`) or **inline** as an `<svg>` element that CSS and JavaScript can style. Inside inline SVG, end every shape with `/>`; an unclosed shape swallows the shapes after it.

The canvas: width, height and viewBox

- `width` and `height` set how big the drawing appears on the page.
- `viewBox="min-x min-y width height"` defines the drawing's own grid of units (its coordinate system): the graph paper every shape is measured on. The browser then maps that grid onto the rendered size.
- The origin (0,0) is the **top-left** corner and y grows **downward**, unlike school graphs.
- Anything outside the viewBox is clipped: silently not drawn, no error.
- `preserveAspectRatio` (default `xMidYMid meet`) decides how the grid is fitted and centred when the proportions differ.

```
<svg viewBox="0 0 200 100" width="400">
  <rect x="0" y="0" width="100" height="50"/>
</svg>
```

200 units are spread across 400 px, so 1 unit = 2 px and the 100-unit rectangle shows 200 px wide. The numbers inside never change; only the scale does.

Trap: viewBox holds no colours, no outline width and no timing; it only sets up the coordinate grid. Colours come from fill and stroke.

The basic shapes and their attributes

Every shape has its own vocabulary. Ask first: is it measured from a corner, from a centre, or from two endpoints?

Element	Positioned by	Size / shape	Remember
`<rect>`	x, y = top-left corner	width, height; rx, ry round the corners	rx here is a corner radius, not a position
`<circle>`	cx, cy = centre	r = radius	x and y mean nothing to a circle
`<ellipse>`	cx, cy = centre	rx = horizontal radius, ry = vertical radius	rx equal to ry gives a perfect circle; it takes no r
`<line>`	x1, y1 = start; x2, y2 = end	no area at all	needs a stroke to be seen
`<polyline>`	points = list of x,y pairs	any number of points	stays open
`<polygon>`	points = list of x,y pairs	any number of points (3 = triangle)	closes back to the first point
`<text>`	x, y = start of the baseline	font-size	a label inside a drawing

```
<rect x="20" y="10" width="80" height="40" rx="8"/>
<circle cx="150" cy="30" r="20"/>
<ellipse cx="220" cy="30" rx="30" ry="15"/>  <!-- wide oval -->
<ellipse cx="290" cy="30" rx="15" ry="30"/>  <!-- tall oval -->
<ellipse cx="360" cy="30" rx="20" ry="20"/>  <!-- perfect circle -->
```

The rectangle's top-left corner sits at (20,10), so its bottom-right corner is at (100,50). The circle is placed by its centre with cx and cy, and sized by its radius r. The three ellipses show the rule: rx bigger = wide oval, ry bigger = tall oval, equal radii = a perfect circle.

Wrong vocabulary is not an error message; the browser simply ignores attributes a shape does not understand. A circle written with x and y keeps its default centre (0,0), so only a quarter of it peeps out of the top-left corner. A rect given cx, cy and r has no width or height, so nothing appears. An ellipse ignores r; without rx and ry it has no size.

Trap: there is no `<triangle>`, `<oval>` or `<arc>` element. A triangle is a polygon with three points, an oval is an ellipse, an arc or curve is a path.

Why a line needs a stroke: a `<line>` has no inside (no area), so fill has nothing to colour, and the default stroke is none. A line with no stroke therefore paints nothing. The markup is still perfectly valid (nothing rejects it), lines do not default to white, and the stroke never decides the length: the endpoints x1, y1, x2, y2 do.

```
<line x1="10" y1="80" x2="250" y2="80"/>                <!-- invisible -->
<line x1="10" y1="80" x2="250" y2="80" stroke="teal"/>  <!-- visible -->
```

polyline versus polygon: both take the same points list and draw straight segments only (neither can curve; curves need a path), and neither is limited to three points. The single difference: polygon automatically joins the last point back to the first; polyline leaves the end open.

```
<polyline points="10,90 50,10 90,90" fill="none" stroke="red"/>  <!-- open ^ -->
<polygon  points="10,90 50,10 90,90" fill="none" stroke="red"/>  <!-- closed triangle -->
```

Give a polyline `fill="none"`: the default fill is black, which would colour the wedge between its points.

Painting: fill and stroke

Memory hook: **fill** is paint poured inside; **stroke** is the pen line around the edge. So fill paints the interior and stroke paints the edge line, never the other way round. Neither sets size, transparency or rotation. Both take any CSS colour; `fill="none"` makes a shape hollow.

Attribute	Controls	Default / example
fill	Interior colour	default black; none = hollow
stroke	Outline colour	default none (no outline)
stroke-width	Outline thickness	stroke-width="4"
stroke-dasharray	Dash pattern: dash length, then gap length, repeated	"12,4" = 12-unit dashes with 4-unit gaps

stroke-linecap	Shape of a line's ends	butt (default, flat at the endpoint), round, square
opacity	Transparency	0 (invisible) to 1 (solid)

```
<circle cx="60" cy="60" r="40"
        fill="none" stroke="crimson"
        stroke-width="3" stroke-dasharray="12,4"/>
```

A hollow ring whose outline is 12 units of colour, then 4 units of gap, repeated all the way round. The thickness (3) comes from stroke-width, not from the dash numbers.

Trap: two numbers in stroke-dasharray are perfectly legal; the value is a list. "10" alone gives 10 on, 10 off; "2,8" looks dotted. The numbers never set thickness and never set a dot radius.

The path language: M, L, H, V, Q, Z

`<path>` can draw anything. Its `d` attribute is a list of commands, each a letter followed by numbers.

Command	Meaning
M x y	Move the pen to (x, y) without drawing; this starts a new shape
L x y	Straight line to (x, y)
H x / V y	Horizontal line to x / vertical line to y
Q cx cy x y	Curve to (x, y), bent toward the control point (cx, cy)
Z	Close: a straight segment back to where the current shape began (the last M point)

Uppercase letters are absolute: "go to this point on the grid". Lowercase letters are relative: each number is an offset from where the pen is now. Lowercase is valid, is used in a path's d attribute, and normally lands somewhere different from the uppercase version.

```
<path d="M10 90 L50 10 L90 90 Z"/>    <!-- absolute -->
<path d="M10 90 l40 -80 l40 80 z"/>   <!-- relative: same triangle -->
```

From (10,90), `l40 -80` means 40 right and 80 up, landing on (50,10). Then `l40 80` lands on (90,90), and z seals the third side back to the M point (10,90).

Trap: Z has nothing to do with zooming, erasing the last command or layering. Z draws a straight segment back to where the current shape began (the last M point), sealing the outline. In a path with a single M, that is simply the starting point. Layering in SVG comes from source order: later elements paint over earlier ones.

Text, groups and accessibility

- `<text x="40" y="90">Lab 3</text>` places a label; `<g>` groups shapes so they move or are styled together.
- Inline SVG has no alt attribute: give it `role="img"` plus `aria-label`, or `aria-hidden="true"` if purely decorative.

CHECK YOURSELF

5.1 A notification dot of radius 12 must have its middle exactly at the point (30, 45). Which tag draws it?

a. <circle x1="30" y1="45" r="12" />
b. <rect x="30" y="45" width="12" height="12" />
c. <circle cx="30" cy="45" r="12" />
d. <circle cx="30" cy="45" radius="12" />

5.2 A page divider must run from (10, 60) to (190, 60) and actually be visible. Which tag works?

a. <line x="10" y="60" width="180" stroke="gray" />
b. <line x1="10" y1="60" x2="190" y2="60" fill="gray" />
c. <line points="10,60 190,60" stroke="gray" />
d. <line x1="10" y1="60" x2="190" y2="60" stroke="gray" />

5.3 An icon is coded as <svg viewBox="0 0 50 50" width="200" height="200"> containing <circle cx="25" cy="25" r="20" />. How wide does the circle appear on screen?

a. 40 px
b. 160 px
c. 200 px
d. 20 px

5.4 Two shapes both use points="10,10 60,10 60,60" with fill="none" and stroke="black": one is a <polyline>, the other a <polygon>. What do you see?

a. The polyline shows two joined segments with an open end; the polygon adds a third side back to (10,10)
b. Both show exactly the same closed triangle
c. The polygon shows nothing, because a polygon needs at least four points
d. The polyline bends smoothly through the points while the polygon uses straight edges

5.5 What does <path d="M 20 20 h 40 v 40 z" fill="none" stroke="black" /> draw?

a. Nothing, because lowercase commands are not allowed in a path
b. A 40 by 40 square
c. A closed triangle with corners at (20,20), (60,20) and (60,60)
d. Two lines forming an open corner, with no side joining the last point to the first

5.6 A sticker needs a hollow circle with an orange dashed outline made of 5-unit dashes and 3-unit gaps. Which attribute set achieves it?

a. fill="orange" stroke="none" stroke-dasharray="5,3"
b. fill="none" stroke="orange" stroke-dasharray="5,3"
c. fill="none" stroke="orange" stroke-width="5,3"
d. fill="orange" stroke="orange" stroke-dasharray="3,5"

6 Multimedia: audio, video and iframes

Textbook chapter 10

Native media in HTML5

Before HTML5, playing sound or film needed a plug-in such as Flash. Now the browser plays media itself and supplies the play button, volume and progress bar. `<audio>` and `<video>` follow one pattern:

- The element itself is **paired**: it always needs its closing tag.
- One or more `<source>` children (void) point at the media files.
- Optional `<track>` children (void) add captions.
- Text between the tags is shown only by a very old browser that does not understand `<audio>` / `<video>` at all. If no source format is playable, a modern browser shows an empty player instead, not the text. Audio and video have no alt attribute, and there is no `<fallback>` element.

```
<audio controls preload="none">
  <source src="anthem.mp3" type="audio/mpeg">
  <source src="anthem.ogg" type="audio/ogg">
  Sorry, your browser cannot play this audio.
</audio>
```

For audio that carries information (a lecture, an announcement), also provide a transcript.

The playback attributes

Attribute	What it does	Watch out
controls	Shows the browser's own play/pause, seek and volume bar	Boolean: controls="false" still shows it; remove the attribute to hide it
autoplay	Starts playing as soon as it can	Usually blocked when it would make sound; allowed when muted
muted	Starts with the sound off	The partner that autoplay needs
loop	Starts again from the beginning every time it reaches the end	No repeat or replay attribute exists
poster	A still picture displayed in the player until the viewer presses play	video only
preload	Hint of how much to fetch early: none, metadata or auto	A hint, not a command
width, height	Displayed size in pixels	video only (audio has no size attributes)

src	A single file instead of source children	Use source for several formats

controls is the attribute that draws the native player. Without it the visitor sees no buttons and has no way to start playback (only a script could). autoplay starts playback, muted silences it, preload hints at downloading: none of them draws any buttons.

```
<video controls width="480" poster="fest-still.jpg">
  <source src="fest.mp4" type="video/mp4">
  Your browser cannot play video.
</video>
```

Until someone presses play, the box shows fest-still.jpg as a cover image. The poster does not start playback (that is autoplay), does not add subtitles (that is track), and does not name a downloaded file (that is the download attribute of a link).

Trap: repeating a clip is `loop`. "repeat" and "replay" are not HTML attributes, and autoplay only starts the clip once. A silent background clip is `<video autoplay muted loop>`: browsers block sound that starts on its own, so autoplay without muted usually fails.

source and track

Browsers do not all decode the same formats. Several `<source>` children are a shortlist, not a playlist: the same film saved in different formats. The browser reads them top to bottom and plays the earliest one in the list that it is able to decode, ignoring the rest. The `type` attribute lets it skip an unplayable file without downloading it.

```
<video controls>
  <source src="trailer.webm" type="video/webm">
  <source src="trailer.mp4"  type="video/mp4">
</video>
```

Trap: the sources never play together or one after another, extra sources do not sharpen the picture, and there is no minimum count: one source (or a plain src) is valid. MP4 plus WebM simply covers essentially every browser.

`<track>` attaches timed text from a WebVTT (.vtt) file: `kind` (subtitles, captions, descriptions, chapters), `srclang`, `label` (the name in the player menu) and `default`. It is void and goes after the sources. Subtitles never come from `<caption>` (tables) or `<source>` (media files), and there is no subtitle element.

```
<track src="talk-en.vtt" kind="captions" srclang="en" label="English" default>
```

iframe: a window into another page

An inline frame embeds a whole separate document, with its own URL, inside yours: a map, a video player, a payment form, a dashboard. What loads inside it is not part of your page's DOM.

```
<iframe src="https://maps.example.org/embed?q=library"
        width="600" height="260"
        title="Map showing the central library"
        loading="lazy"></iframe>
```

The width and height attributes take whole numbers of pixels only. For a full-width frame use CSS, for example the stylesheet rule `iframe { width: 100%; }`, not width="100%".

Attribute	Purpose
src	Address of the page to embed
width, height	Displayed size in pixels (there are no rows/cols or x/y attributes)
title	Names the frame for screen-reader users
loading="lazy"	Waits until the frame is near the screen before loading it
allow	Grants powers: fullscreen, camera, microphone, geolocation, autoplay
sandbox	Removes powers; tokens such as allow-scripts or allow-forms give single ones back

Why every iframe needs a title: a screen reader announces the title when the user reaches the frame, so they know what is inside ("Map showing the central library"). Without it they hear only "frame". Nothing is printed above the frame, the frame loads without it, and picture quality is unaffected.

Embedding responsibly

Embedding is a trade. The other site's scripts run in your visitor's browser, so a third-party frame can drop cookies, follow your visitors from site to site, and make your page load more slowly. Habits that limit the damage:

- Embed only from trusted sources, and add `loading="lazy"` so off-screen frames cost nothing.
- Use `sandbox` to grant only what the widget needs. An empty sandbox blocks scripts, forms and pop-ups, so a widget that dies after you add it has lost its scripts: add `allow-scripts`.
- Use `allow` to switch on fullscreen or camera only where needed.
- A big site that shows a blank frame is refusing to be framed (an X-Frame-Options or Content-Security-Policy header). Nothing in your HTML overrides it; use the site's official embed address, such as a video site's /embed/ link.

Trap: an iframe leaves your own stylesheet working, leaves the browser's Back button working, and keeps your page as normal HTML. The genuine costs are privacy (cookies, tracking) and speed.

CHECK YOURSELF

6.1 A course-intro clip must show a still photo of the lecturer until someone starts it, and visitors must get play and pause buttons. Which tag does this?

a. <video src="intro.mp4" autoplay thumbnail="lecturer.png"></video>
b. <video src="intro.mp4" poster="lecturer.png" preload="auto"></video>
c. <video src="intro.mp4" controls poster="lecturer.png"></video>
d. <video src="intro.mp4" controls track="lecturer.png"></video>

6.2 A gym's background clip has loop, then <source src="gym.webm" type="video/webm"> followed by <source src="gym.mp4" type="video/mp4">. In a browser that can play both formats, what happens?

a. It plays gym.webm and restarts it from the beginning each time it finishes
b. It plays gym.webm, then gym.mp4, then stops
c. It plays both files side by side at the same time
d. It always plays gym.mp4, because MP4 has priority over WebM

6.3 A screen-reader user moving through a page reaches an embedded weather widget and hears only "frame". What is the right fix on the <iframe>?

a. Add alt="Weather widget"
b. Type the words Weather widget between <iframe> and </iframe>
c. Add loading="lazy"
d. Add title="Live weather for the campus"

UNIT 2

Semantic structure, accessibility, forms and tables

7 Semantic elements and page structure

Textbook chapters 13, 14

What "semantic" means

Definition. A semantic element is one whose name describes the meaning and purpose of its content. `<nav>` says "these are navigation links"; `<footer>` says "this is closing information". A non-semantic element such as `<div>` or `<span>` says nothing about its content: it is just a box.

On screen, `<div class="header">` and `<header>` look the same: both are plain block boxes. The difference is for machines:

- **Screen readers** announce semantic regions as **landmarks**, so a blind user jumps with one key to the main content, menu or footer. On an all-div page they must hear the whole menu on every visit.
- **Search engines** tell real content apart from repeated boilerplate.
- **Developers** read the page structure from the tag names alone.

```
<!-- just boxes: no meaning -->
<div class="menu">...</div>
<div class="content">...</div>

<!-- same look, now a map machines can read -->
<nav>...</nav>
<main>...</main>
```

Semantic means **meaning, not looks**. The gain is machine-readable structure for screen readers and search engines.

Trap: Semantic elements do **not** add a special colour or style, do **not** render or load faster than a `<div>`, and do **not** make the file automatically smaller. Most of them (`<header>`, `<nav>`, `<section>`, `<article>`, `<aside>`, `<footer>`) may appear many times; the one-per-page rule applies only to `<main>`.

The seven page-structure elements

Element	What it means	How many	Typical content
`<header>`	Introductory content of the page or of a section or article	One for the page; more inside articles and sections	Logo, site title (the h1), tagline, often the main menu
`<nav>`	A major group of navigation links	One per group	Main menu, breadcrumbs, page numbers, table of contents
`<main>`	The dominant content unique to this page	One per page (never more than one visible)	The sections, articles and related asides
`<section>`	A thematic group of content with its own heading	Many	"Our services", "Results", "FAQ"
`<article>`	Self-contained content that could be distributed on its own	Many	News story, blog post, product card, user comment
`<aside>`	Related but not essential; the page still makes sense without it	Many	Related-articles sidebar, pull quote, advert
`<footer>`	Closing information for the page or a section or article	One for the page; more inside articles and sections	Copyright, contact details (in `<address>`), author info

Rules for main: A page should have one main element, and never more than one visible main (extra ones must carry the `hidden` attribute). It must not sit inside `<article>`, `<aside>`, `<header>`, `<footer>`, `<nav>` or `<section>`, because it is the dominant content of the whole document. Site-wide repeats (logo, menu, copyright) stay outside it. By contrast, `<section>`, `<article>` and `<div>` may appear as often as the content needs.

Trap: `<header>` is not `<head>`. `<head>` holds invisible metadata (charset, title, CSS links); `<header>` is visible introductory content inside `<body>`. `<title>` is only the text on the browser tab, and HTML has no `<top>` element at all.

nav: wrap the whole group once

`<nav>` marks a navigation region, not a single link. Put one `<nav>` around the whole group, usually around a list of links. Main menus, breadcrumbs, pagination links and a table of contents are each a group of links, so each gets its own `<nav>`. When a page has two or more, give each an `aria-label` so a screen reader can tell them apart.

```
<nav aria-label="Breadcrumb">
  <ol>
    <li><a href="/">Home</a></li>
    <li><a href="/shoes/">Shoes</a></li>
    <li><a href="/shoes/running/" aria-current="page">Running</a></li>
  </ol>
</nav>
```

Trap: One `<nav>` per link is wrong: nav marks the group, and a dozen tiny landmarks make the screen reader repeat "navigation". Also false: "nav may hold only images" (it holds lists, text, anything) and "nav cannot go inside header" (the main nav inside the page header is normal). `<menu>` is not navigation: in HTML it is a toolbar-style list of commands with no navigation landmark. A link group does not turn into an `<aside>` or a `<header>` because of where it sits.

section, article, aside or div?

Ask these questions in order:

1. Would the block still make sense on another website or in a news feed? `<article>`.
2. Related to the main content but removable without loss? `<aside>`.
3. A themed part of this page with its own heading? `<section>`.
4. None of these; it exists only for CSS? `<div>`.

Content	Element	Why
A recipe that a food blog could republish whole	`<article>`	Stands alone
The "Our menu" block of a cafe home page, with an h2	`<section>`	Themed part of this page
A "You may also like" column beside a story	`<aside>`	Related, not essential
A wrapper only so CSS can make two columns	`<div>`	No meaning to convey
Copyright line and phone number at the bottom	`<footer>`	Closing information
Breadcrumbs, main menu, page numbers	`<nav>`	Groups of navigation links

A "related stories" sidebar is an `<aside>`: extra content next to the story. It is not the site's navigation (`<nav>`), not the closing information (`<footer>`) and not the introduction (`<header>`).

section vs article in one line: `<section>` groups thematically related content of this page; `<article>` is complete enough to be distributed independently. They are not the same, and neither is limited to one per page. An article may contain sections, and a section may contain articles.

Every section should have a heading. The heading lets assistive technology summarise the section and lets users jump between sections by heading. The page renders and takes CSS perfectly well without one; the heading is for meaning, not display. If a block has no natural heading, it was probably never a section: use a `<div>`.

div is still right as a generic styling wrapper with no meaning of its own: a grid container, a card shell, a box for a background colour. It is the wrong choice for anything a semantic element already describes, such as a copyright notice (footer), the main menu (nav) or a blog post (article).

Trap: The two definitions are often swapped. It is **article** that is independently distributable and **section** that is a thematic group. A missing heading does not stop a section from rendering, does not block CSS and does not hide it from search engines; it only makes the section harder to navigate.

Assembling the page skeleton

A page is a tree: `<html>` holds the invisible `<head>` and the visible `<body>`; the body holds the regions, in reading order; CSS arranges them into columns later.

```
<body>
  <header><h1>City Library</h1> <nav>...</nav></header>
  <main>
    <section><h2>New arrivals</h2>
      <article><h3>Review: a mystery novel</h3>...</article>
    </section>
    <aside><h2>Events this week</h2>...</aside>
  </main>
  <footer>&copy; 2026 City Library <address>12 Mall Road</address></footer>
</body>
```

- **Page header:** logo or site title and usually the main nav (header > nav).
- **Where articles and asides go:** in the usual skeleton they sit inside the main region. Never inside `<head>` (metadata only) and not inside the page header or footer. (A site-wide sidebar placed beside main is also valid, but asides about the page content belong in main.)
- **Page footer last:** copyright, contact details in `<address>`, small print. This is closing information, not a related extra (aside), not a link group (nav) and not a themed block (section).
- **Repeats:** `<header>` and `<footer>` may appear again inside an article (byline on top, tags below). Only main is limited: never more than one visible main per page.
- **Headings follow regions:** one h1 in the page header, an h2 per section, an h3 per article.

CHECK YOURSELF

7.1 A cinema website is rebuilt. Version 1 uses and ; version 2 uses <nav> and <main>. Both look identical in a browser. What does a blind visitor using a screen reader gain from version 2?

a. The browser automatically adds a drop-down animation to the menu
b. She can jump straight to the menu or the film listings using landmark shortcuts
c. Version 1 is never indexed by search engines, so only version 2 appears in results
d. Version 2 reads itself aloud even on a computer with no screen reader installed

7.2 A recipe page needs a box whose only job is to let CSS place the ingredient list and the cooking-tips column side by side. Which element should wrap the two columns?

a. <figure>, because the two side-by-side columns are shown together like one illustration
b. <address>, because the wrapper gives each of the two columns its own fixed place
c. A second <main>, so that the ingredients and the tips each count as main content
d. , because this wrapper exists only for the layout and carries no meaning

7.3 A train-booking site shows page-number links (1, 2, 3, Next) under its search results. Which markup is correct?

a. One <nav aria-label="Result pages"> around a single list holding all four links
b. A separate <nav> around each of the four links, so every page number is its own landmark
c. A <footer> around the links, because they appear after the results
d. A plain <ol> with no wrapper, because any list already counts as a navigation landmark

7.4 A sports news page about a cricket series contains: (P) a match report that a partner site could republish unchanged, (Q) a block headed "Squad list" that only makes sense on this page, and (R) a side box of "Other cricket stories". Which mapping is best?

a. P = section, Q = article, R = nav
b. P = aside, Q = section, R = article
c. P = article, Q = section, R = aside
d. P = article, Q = footer, R = header

7.5 Which skeleton for a hotel's home page is structured correctly?

a. <main> wraps everything, logo, menu, room cards and copyright, because all of it is content on this page
b. <header> holds the logo and menu; <main> holds the room cards and a "Nearby sights" <aside>; <footer> holds the phone and copyright
c. <aside> holds the logo and menu, since they repeat on every page; <main> holds the room cards; <footer> holds the phone and copyright
d. <header> holds the logo and menu; <main> holds the room cards; the phone and copyright go last in a <section> that has no heading

7.6 A developer deletes the <h2> from every <section> on a museum's opening-hours page. The page still looks almost the same. What has actually been lost?

a. Nothing important: a heading inside a section only sets the font size of its first line
b. The sections stop rendering in older browsers, so visitors see blank areas instead
c. Screen-reader users lose the labels they use to identify and jump between the sections
d. Search engines are now blocked from indexing every section that has no heading

8 Accessibility: POUR, WCAG, ARIA and keyboard access

Textbook chapters 15, 20

What web accessibility is

Web accessibility is the practice of building sites that people with visual, auditory, motor or cognitive disabilities can perceive, understand, navigate and use. In short: nobody is locked out.

- **Visual:** blind users hear the page through a screen reader (NVDA, JAWS, VoiceOver, TalkBack); low-vision users zoom or magnify.
- **Auditory:** deaf users need captions and transcripts.
- **Motor:** some users cannot hold a mouse and use only the keyboard, a switch or voice control.
- **Cognitive:** clear language, consistent menus and helpful error messages.

It helps everyone: captions on a noisy bus, good contrast in sunlight, keyboard shortcuts for power users. Most of it is simply correct HTML: `lang`, real headings, `alt` text, labels, landmarks and native controls.

> **Trap:** Accessibility is not a speed target, not a search-ranking trick and not "works only in the newest browser". A fast, top-ranked page can still be unusable for a blind or keyboard-only visitor.

WCAG and its conformance levels

WCAG stands for Web Content Accessibility Guidelines, the international standard published by the W3C through its Web Accessibility Initiative (WAI). Versions: 2.0 (2008), 2.1 (2018), 2.2 (2023); each later version adds criteria to the earlier one (2.2 also dropped one obsolete rule, 4.1.1 Parsing). Structure: 4 principles (POUR), then guidelines, then testable **success criteria**, each given a level.

Level	Meaning	Example criteria
A	The minimum; without it some users cannot use the page at all	Images have text alternatives; everything works by keyboard; captions on recorded video
AA	The usual target required by most laws and organisations	Normal text contrast at least 4.5:1; visible focus; captions on live video
AAA	The highest; not required for whole sites because some content cannot meet it	Contrast 7:1; sign-language video

Levels are **cumulative**: AA means every A and every AA criterion passes. When a policy says "WCAG 2.x compliant" it almost always means Level AA. India's RPwD Rules 2017 require government sites to follow GIGW, which adopts WCAG 2.1 Level AA; the US Section 508 and the EU's EN 301 549 also point to WCAG Level AA.

> **Trap:** Made-up expansions built from words like "World", "Code", "Compliance", "Grading" or "Generator" are wrong: it is **Web Content Accessibility Guidelines**. There are only three levels, A, AA and AAA; there is no Level B. Level A is only the floor, and AAA is the ideal, not the normal legal target.

POUR: the four principles

Principle	The question it asks	Typical checks	A scenario that tests it
Perceivable	Can users sense the content?	alt text, captions, transcripts, contrast	A blind user hears a product photo described
Operable	Can users work every control?	Keyboard-only use, visible focus, skip links, enough time, nothing flashing more than 3 times a second	Finishing a checkout with only Tab and Enter
Understandable	Can users understand the content and the controls?	Page language declared, consistent navigation, labels, clear error messages	A form says "Enter a 6-digit PIN code" instead of just "Invalid"
Robust	Does it keep working across browsers, devices and assistive technologies?	Standard HTML, native elements, correct name, role and value for every control	The same form works in Chrome, Firefox, on a phone and with a screen reader

Scenario trick: **sensing** = Perceivable; **doing, pressing keys** = Operable; **making sense** = Understandable; **works everywhere, with any tool** = Robust.

Trap: A keyboard-only test is about operating the page, so it is **Operable**, not Perceivable (nothing is hidden from the senses) and not Understandable (the user already knows what to do). "Keeps working across browsers, devices and assistive technology" is **Robust**, not Operable.

Keyboard access and tabindex

Many users never touch a mouse. **Tab** moves to the next control, **Shift+Tab** to the previous one, **Enter** follows a link or presses a button, **Space** toggles a checkbox or presses a button, and the **arrow keys** move inside radio groups and menus. Native links (`<a href>`), buttons and form controls are focusable and keyboard-operable for free; a `<div onclick>` is not. Never remove the focus outline without a replacement. A skip link at the top (`<a href="#main">Skip to main content</a>`) lets keyboard users bypass the menu.

Value	Effect
`tabindex="0"`	Element becomes focusable and joins the normal Tab order at its position in the source
`tabindex="-1"`	Not reachable with Tab, but a script can focus it (skip-link target, dialog)
`tabindex="3"` (any positive number)	Jumps the queue and scrambles the order; avoid

```
<div class="seat" tabindex="0">Seat 14B</div>
<!-- Tab now stops here, in source order.
     Enter and Space still need a script. -->
```

Trap: `tabindex="0"` does **not** move an element to the front of the Tab order (zero means "normal order"), does **not** remove it from the order (that is `-1`), and does **not** hide anything from screen readers (that is `aria-hidden`).

ARIA: role, aria-label, aria-hidden and the first rule

ARIA (WAI-ARIA, Accessible Rich Internet Applications) is a W3C set of HTML attributes that tell assistive technology what an element is, what it is called and what state it is in, where plain HTML cannot say it. ARIA changes **what is announced**, never **how the element behaves**: `role="button"` on a `<div>` makes a screen reader say "button", but the div still ignores Enter and Space and is not in the Tab order.

First rule of ARIA: if a native HTML element already has the meaning and behaviour you need, use it instead of rebuilding it with ARIA. A real `<button>` beats `<div role="button">`. Adding `role="button"` to a `<button>` is redundant, and giving a native element a role that contradicts it is worse. ARIA is a fallback for gaps, not a newer replacement for semantic HTML.

Attribute	What it gives	Example
`role`	Purpose of a non-semantic element for assistive technology	`<div role="alert">Saved</div>`

`aria-label`	Accessible name written directly in the attribute; not shown on screen	`<button aria-label="Close">×</button>`
`aria-labelledby`	Name taken from another element's id	`<section aria-labelledby="t1"><h2 id="t1">Fees</h2>`
`aria-describedby`	Extra description, read after the name; its value is an id, not text	A password rule shown under an input
`aria-hidden="true"`	Hides the element from assistive technology; it stays visible on screen	A decorative icon beside a labelled button

```
<button aria-label="Search"><svg aria-hidden="true">...</svg></button>
<button><svg aria-hidden="true">...</svg> Print</button>
```

The first button has no visible text, so `aria-label` names it "Search". In the second, the word Print already names the button, so the decorative printer icon is hidden to stop it being read as noise.

> **Trap:** `aria-label` is not a visible caption (sighted users never see it), not a validation rule (that is `required` or `pattern`) and not a CSS hook (that is `class`). `aria-hidden="true"` must never go on content people need: the main heading, a form field, a button's own text, or anything focusable. `role` does not set the Tab order (that is `tabindex`), does not style anything, and does not restrict who may see an element; it only says what the element is.

CHECK YOURSELF

8.1 An audit of a bank's website finds four problems: (W) a promo video has no captions; (X) a date picker opens only on a mouse click; (Y) a form error says only "Error 37"; (Z) a custom slider works in one browser, but screen readers cannot report its value. Which principle does each break?

a. W Operable, X Perceivable, Y Robust, Z Understandable
b. W Perceivable, X Robust, Y Understandable, Z Operable
c. W Understandable, X Operable, Y Perceivable, Z Robust
d. W Perceivable, X Operable, Y Understandable, Z Robust

8.2 A tender for a city council portal says the site must be "WCAG 2.2 AA compliant". What must the developers satisfy?

a. Only the AA success criteria; Level A criteria can be skipped because AA replaces them
b. Every Level A and every Level AA success criterion of the Web Content Accessibility Guidelines
c. Only the colour-contrast rules, because AA is the contrast grade of the guidelines
d. Every Level AA criterion plus any AAA criterion that applies to the page's content

8.3 A video player has a Mute button that shows only a speaker icon drawn as an <svg>, and a Share button that shows the visible word Share followed by a decorative arrow <svg>. Which markup is correct for screen-reader users?

a. <button aria-hidden="true"><svg>...</svg></button> and <button>Share <svg aria-hidden="true">...</svg></button>
b. <button aria-label="Mute"><svg aria-hidden="true">...</svg></button> and <button>Share <svg aria-hidden="true">...</svg></button>
c. <button aria-label="Mute"><svg aria-hidden="true">...</svg></button> and <button>Share <svg>...</svg></button>
d. <button aria-describedby="Mute"><svg>...</svg></button> and <button aria-hidden="true">Share <svg>...</svg></button>

8.4 In a photo gallery each thumbnail is a that opens a larger view when clicked. The developer adds tabindex="0" to every thumbnail. What changes?

a. Each thumbnail can now be reached with Tab, in the same order as it appears in the source
b. Screen readers now announce each thumbnail as a button
c. The thumbnails are moved to after every native link and button in the Tab sequence
d. Pressing Enter on a thumbnail now opens the larger view without any extra script

8.5 A shop needs a control that submits its order form. Which choice follows best practice?

a. Place order , since the class name tells screen readers what it is
b. Place order, since the role makes it respond to Enter and Space
c. <button type="submit">Place order</button>, since it is focusable and submits on Enter or Space with no extra code
d. <a>Place order with no href, since CSS can make it look and act like a button

9 Forms, input types and validation

Textbook chapters 16, 17, 18

The form element: where and how the data travels

A `<form>` collects what the user enters and sends it to a server. When the submit button is pressed, the browser takes every control that has a `name`, pairs that name with the current value (`name=value`) and sends the pairs. Exceptions: disabled controls, unticked checkboxes and unchosen radios send nothing, and only the button that was pressed is included. Two attributes on the form decide the journey:

- **action** = *where*: the URL the form data is sent to. Leave it out and the form submits to the current page.
- **method** = *how*: `get` (the default) or `post`.

```
<form action="/library/renew" method="post">
  <input type="hidden" name="book" value="B-204">
  <button>Renew</button>
</form>
```

Result: `book=B-204` goes to `/library/renew`, inside the request.

	method="get"	**method="post"**
Where the pairs travel	Appended to the URL as a query string: `/search?q=maps&page=2`	In the body of the HTTP request; the URL stays clean
Shows in address bar and history; can be bookmarked	Yes	No
File uploads	No	Yes, with `enctype="multipart/form-data"`
Use for	Searches, filters	Logins, registrations, orders, uploads

Trap: POST is *not* encryption. On a plain `http://` page anyone watching the network can read the body; only HTTPS encrypts. POST carries files as well as text and does not switch off validation. `action` is only the destination address: never a validation rule, the method or a button name.

name, id, value and the label

Attribute	**Who uses it**	**Job**
`name`	The server	The key the value is submitted under. No name, no data.
`id`	The page	Unique on the page; what a label's `for` points at. Never sent.
`value`	Both	The data: pre-filled text, or what a chosen radio, checkbox or option sends.
`class`	CSS	Styling only; it links nothing.

A label is linked to its control when the label's for value matches the control's id exactly (not its name, class or type). Wrapping the control inside the label also works. Then clicking the label focuses the box, and screen readers read the two together.

```
<label for="roll">Roll number</label>
<input type="text" id="roll" name="rollno">
```

The label finds the box through `id="roll"`; the server receives `rollno=...`.

Placeholder is a hint, not a label. It disappears as soon as typing starts, so the user can forget what the box was for, and screen readers may not announce it. Keep a visible `<label>` and use placeholder only for a sample, e.g. `placeholder="e.g. 98765 43210"`.

Trap: placeholder is not deprecated (it arrived with HTML5), it *can* be styled with CSS (`::placeholder`), and it neither blocks nor allows submission: it is not a value at all.

Choosing the right control

The user must...	Use	Not
pick exactly one of a few (one T-shirt size)	radio buttons that share one name	checkbox (allows many)
tick any number of independent options	`<input type="checkbox">`	radio
pick several items from one list	`<select multiple>` (Ctrl/Shift-click)	plain `<select>` (one only)
write several lines (a complaint, an address)	`<textarea>`	`<input>` (one line only)

```
<fieldset>
  <legend>T-shirt size</legend>
  <label><input type="radio" name="size" value="s"> S</label>
  <label><input type="radio" name="size" value="m"> M</label>
  <label><input type="radio" name="size" value="l"> L</label>
</fieldset>
```

The **shared name** is what makes radios one group: choosing one un-chooses the others, and only the chosen one's value is sent (`size=m`). The `id` cannot be shared (it must be unique), `class` only styles, and each `value` should differ, because the value is what gets sent as the answer.

Trap: give each radio a different name and each becomes its own group of one, so all of them can be selected at once.

```
<label for="msg">Your complaint</label>
<textarea id="msg" name="msg" rows="4"></textarea>

<label for="labs">Labs you will attend</label>
<select id="labs" name="labs" multiple>
  <option>Physics</option><option>Chemistry</option><option>Workshop</option>
</select>
```

Trap: `<input type="textarea">` and `type="multiline"` do not exist. The textarea is its own element with a closing tag, and default text goes between the tags. A `<select>` is a dropdown of fixed choices, not a typing box.

Input types worth knowing

One element, `<input>`, becomes a different control for each `type`.

type	What the user gets
`text`	A single-line box (the default)
`password`	Typed characters are masked as dots
`hidden`	Nothing on screen at all; the value is still sent
`email`	Address format is checked; phones show an @ key
`number`	A box with an up/down spinner; takes min, max, step
`range`	A slider for a value between min and max
`date`	A native calendar picker
`time`, `month`, `week`, `datetime-local`	Other native pickers (some desktop browsers show a plain text box for month and week)
`color`, `file`	Colour picker; file chooser

Trap: made-up types such as `secret`, `mask`, `slider`, `scale` or `calendar` raise no error: the browser quietly treats them as `text`, so a "secret" box shows the password in plain view. And `hidden` is not masking: it hides the whole field.

Buttons: which one submits?

```
<button type="submit">Book seat</button>  <!-- sends the form -->
<button type="reset">Clear</button>       <!-- restores starting values -->
<button type="button">Show map</button>   <!-- does nothing without JavaScript -->
```

- **No type written?** Inside a form, a button with no type is a submit button (submit is the default).
- `<input type="submit">` also submits.
- A link such as `<a href="#">` only navigates (here, to the top of the page); it never submits.

> **Trap:** the caption is irrelevant. `<button type="button">Submit</button>` still does nothing, and a reset button sends nothing anywhere.

Built-in validation attributes

HTML5 **constraint validation**: attributes describe a valid value; on submit the browser checks them all. If one fails, it stops the submission, focuses the first bad field and shows a message. No JavaScript needed.

Attribute	Blocks submission when	Example
`required`	the field is empty (no radio chosen, box unticked)	`<input name="city" required>`
`min`, `max`	a number or date is below min or above max	`type="number" min="5" max="40"` accepts 5 to 40
`minlength`	fewer characters than the limit	`minlength="6"`: at least 6 characters of any kind
`maxlength`	never in practice: the browser stops typing at the limit	`maxlength="10"`: at most 10 characters
`pattern` + `title`	the whole value does not match the regular expression	see below
`placeholder`, `value`	never: one is a hint, the other a pre-fill	a pre-filled required field counts as filled

> **Value versus length.** min/max limit how big a number or date is; minlength/maxlength limit how many characters are typed. So "age 21 to 65" is `min="21" max="65"`, while `minlength="8"` on a password means at least 8 characters (not at most, not exactly, not digits only, nothing to do with time). An input has no `start` / `end` or `range="..."` attributes; `range` is a type.

An empty box passes both `minlength` and `pattern`: they only judge a value that is actually there. That is why they are paired with `required`.

```
<label for="sub">Subject code</label>
<input id="sub" name="sub" required pattern="[A-Z]{2}[0-9]{3}"
       title="Two capital letters then 3 digits, e.g. CS101">
```

Type `cs1` and the browser blocks the form; its message shows the title text as the hint about the expected format. The title is not the server's key (that is `name`), not a default value (that is `value`) and no replacement for a `<label>`.

What type="email" really checks

Browsers follow the HTML Living Standard: one or more allowed characters (letters, digits and symbols such as . + _ -, but no spaces), one @, then a domain made of one or more labels of letters, digits and hyphens, separated by dots. A dot after the @ is allowed but not required.

Value in `<input type="email" required>`	Result
`user.k@campus.ac.in`	Valid: dots are fine on both sides of the @
`user@campus`	Valid: the form submits normally
`user@`	Blocked: nothing after the @
`usercampus.ac.in`	Blocked: no @ at all
`user k@campus.in`	Blocked: a space inside

(empty)	Blocked by required

> **Trap:** many notes say an address with no dot after the @ (like `name@college`) is rejected. Real browsers accept it and submit normally. When a value really is invalid, the browser stops the form; it never "corrects" the address and never silently drops the field from the data.

For a stricter rule, add a `pattern`; it is checked on top of the email check:

```
<input type="email" name="mail" required
       pattern=".+@.+\..+" title="Use a full address such as name@college.edu">
```

Now `user@campus` is blocked, because the pattern demands a dot after the @.

Client-side versus server-side validation

Client-side checks are the guard at the gate: quick and polite, but they can be walked past. Server-side checks are the lock on the door.

	Client-side (browser)	**Server-side (backend)**
Runs	in the browser, before anything is sent	on the server, after the data arrives
Written in	HTML attributes, JavaScript	backend code (PHP, Python, Java...), not HTML
Feedback	instant, no reload	only after a round trip, so slower
Can the user bypass it?	Yes: DevTools can delete `required`, or a request can be sent without the form	No: it runs on the site's own machine, out of the user's reach
Purpose	convenience (user experience)	security: the **final line of defence**

> Always do both. Client-side validation improves the experience but never replaces the server check, is not more secure, and does not run on the server.

CHECK YOURSELF

9.1 A hostel portal has <form action="/hostel/apply" method="post">. When the student presses Submit, what happens to the filled-in values?

a. They are added after a ? in the address bar of /hostel/apply
b. They stay on the current page, because post ignores action
c. They are sent to /hostel/apply inside the body of the request
d. They are encrypted by post and sent to the submit button's name

9.2 To save space, a designer deletes every <label> from a sign-up form and keeps only placeholder text inside the boxes. What is the real problem?

a. Browsers refuse to submit a form that uses placeholders
b. The hint vanishes once typing starts and may not be announced
c. The placeholder text is sent to the server instead of the typed value
d. Placeholder text is not allowed on type="email" inputs

9.3 A mess-feedback form offers three radio buttons for meal plan, written with name="veg", name="jain" and name="nonveg". What will a user notice?

a. Choosing one automatically clears the other two
b. The browser turns the three radios into a dropdown
c. The form refuses to submit until the names are fixed
d. All three can be selected at the same time

9.4 A developer writes <input type="secure" name="wifi"> for a Wi-Fi passphrase on a lab computer. What does the person typing see?

a. The passphrase in plain, readable characters
b. A row of dots, because secure is a masked type
c. No box at all, because secure hides the field
d. An error message in place of the form

9.5 On a blog editor, Save must send the post to the server, while Preview must only run a script that opens a pop-up and must never send anything. Which markup does exactly that?

a. <button type="reset">Save</button> <button>Preview</button>
b. <button>Save</button> <button>Preview</button>
c. <button type="button">Save</button> <button type="submit">Preview</button>
d. <button>Save</button> <button type="button">Preview</button>

9.6 A locker PIN field is written <input type="password" name="pin" minlength="6">. The user leaves it completely empty and presses Submit. What happens?

a. Submission is blocked because an empty box has fewer than 6 characters
b. The form is sent, because minlength does not judge an empty box
c. The browser shows an error because minlength works only on type="text"
d. The PIN is left out because password fields are never submitted

9.7 A room-booking box is written <input name="room" pattern="[A-D]-[0-9]{3}" title="Block letter, a hyphen, then the room number, e.g. B-214">. A student types b214 and presses Submit. Which describes what she experiences?

a. The form is sent with b214, because the title only tells the server what the box holds
b. The browser rewrites b214 to B-214 by copying the example in the title
c. Submission is stopped, the box is focused, and the error message includes the title's text about the expected format
d. Submission is stopped silently with no message, because a title is only ever shown on mouse hover

9.8 An online canteen form limits quantity with <input type="number" min="1" max="5">. A user opens DevTools, deletes max and orders 500 plates. What should stop this order?

a. A check in the server code that validates the quantity again
b. Nothing else is needed; the browser restores the deleted max before sending
c. Adding placeholder="max 5" to the field
d. Switching the form from get to post

9.9 Which value would a browser refuse in <input type="email" name="mail" required> (no pattern attribute)?

a. office@hostel
b. office.s@uni.ac.in
c. office@
d. o+notes@mail.com

10 Tables and data representation

Textbook chapter 19

When to use a table, and what each part does

Use a table only for **tabular data**: information whose meaning comes from its position, where a row meets a column. A bus timetable (stop by time), a fee chart (course by year), a results sheet (team by event). If you only want boxes side by side, that is layout, and layout is CSS's job (see the last part of this section).

Element	Its job	It is NOT
`<table>`	The container for the whole grid	a page-layout tool
`<caption>`	A visible title for the whole table, read out first by screen readers. Must be the first child of `<table>`.	a column width, a totals row, or a tooltip on each cell (tooltips come from the `title` attribute)
`<thead>`, `<tbody>`, `<tfoot>`	Sort the rows into sections: heading rows, data rows, and summary rows such as totals	a single row (each group holds tr rows), a way to merge cells, a border, or a shortcut that lets you skip tr
`<tr>`	One horizontal row; every cell must sit inside a row	a column (HTML builds a table row by row; no tag holds a column of cells)
`<th>`	A header cell: names what its column or row contains	just "bold text" (bold and centred is only its default look)
`<td>`	A data cell: holds one value	a heading

Memory hook: **tr** = table **r**ow, **th** = table **h**eader, **td** = table **d**ata. A table holds rows, rows hold cells, cells hold content. The group tags sit between table and tr: they group rows, they never replace them.

Why group rows at all? CSS can style the heading section differently from the body, a long table's heading rows can repeat on every printed page, and screen readers can tell heading rows from data rows.

The correct order, in one example

Children of `<table>` go in this order: `caption`, then `colgroup` (optional), then `thead`, `tbody`, `tfoot`.

```
<table>
  <caption>Canteen prices</caption>
  <thead><tr><th scope="col">Item</th><th scope="col">Rs</th></tr></thead>
  <tbody>
    <tr><th scope="row">Samosa</th><td>15</td></tr>
    <tr><th scope="row">Tea</th><td>10</td></tr>
  </tbody>
  <tfoot><tr><th scope="row">Both</th><td>25</td></tr></tfoot>
</table>
```

Browsers draw no borders by default; add them in CSS (`th, td { border: 1px solid #999; }`), not with the old `border` attribute. If you leave out `<tbody>`, the browser quietly adds one.

Trap: `<table><td>5</td></table>` is invalid: the cell is not inside a `<tr>` (the browser patches it, but the source is still wrong). A caption placed after `<thead>` or outside `</table>` is also wrong.

Merging cells: colspan and rowspan

- `colspan="n"`: the cell stretches **sideways** across n **columns**.
- `rowspan="n"`: the cell stretches **downwards** across n **rows**.

Both go on a `<td>` or `<th>`, and n counts grid cells, not pixels. Like knocking down a wall between two rooms: one bigger room, so the plan stops showing two.

Counting rule: every row must add up to the table's column count, spans included. The row below a rowspan cell leaves that cell out, because the slot is already filled from above.

```
<!-- 3 columns: Department | Lab | Day -->
<tr><th rowspan="2">Physics</th><td>Optics</td><td>Mon</td></tr>
<tr><td>Mechanics</td><td>Thu</td></tr>
<tr><td colspan="3">All labs close at 5 pm</td></tr>
```

Count it. Row 1: Physics + 2 cells = 3. Row 2: only 2 cells are written, because Physics (rowspan) still fills the Department slot: 1 + 2 = 3. Row 3: one cell with colspan="3" = 3. One label (Physics) heading two rows that sit one under the other is exactly the rowspan job. So colspan="2" covers two columns side by side, and rowspan="2" covers two rows top to bottom.

Trap: colspan never means rows, whole tables or pages, and neither attribute takes pixels. `span` is not a merge attribute for cells: it belongs on `<col>` and `<colgroup>`, where `<col span="2">` styles two whole columns at once. `scope` (any value, including rowgroup) only says which cells a header describes; it merges nothing. Under a rowspan cell, do not repeat the cell or add an empty one: either makes the row one column too wide.

`<colgroup>` and `<col>` sit after the caption and before `thead`, and let you give a whole column a width or background without repeating it on every cell.

Tables a screen reader can read

A sighted reader glances up to the column heading and left to the row heading. A screen reader reads one cell at a time, so the markup must say which headings belong to each value. Done right, a cell is announced as "Relay, Blue House, 38", not a bare "38". Three tools:

1. Use th for every heading cell, column and row headings alike; td is only for values.
2. Put `scope` on each `<th>` to say which cells it labels.

3. Add a `<caption>`, so the user hears what the table is about before hearing dozens of numbers.

scope value	The header labels...
`col`	the whole column it heads (every cell under it)
`row`	the whole row it starts
`colgroup` / `rowgroup`	a group of columns (a colgroup) / every row of its thead, tbody or tfoot

```
<table>
  <caption>Sports day: points by house</caption>
  <tr><td></td><th scope="col">100 m</th><th scope="col">Relay</th></tr>
  <tr><th scope="row">Red House</th><td>42</td><td>30</td></tr>
  <tr><th scope="row">Blue House</th><td>35</td><td>38</td></tr>
</table>
```

For complex tables with two levels of headings, give each `<th>` an id and list them on the data cell: `<td headers="relay blue">`. scope handles simple tables; headers handles the rest.

A data table that sounds confusing when read aloud is fixed by its markup, not its looks: `<th>` for headings + `scope` + a `<caption>`.

Trap: bigger fonts, bold text and background colours change only what sighted people see; a screen reader ignores them. Rebuilding the grid from `<div>`s throws away the row and column links entirely. Sprinkling colspan merges cells and wrecks the grid without adding a single heading. And scope="col" labels every cell in its column, not just the one cell beneath the heading and not the whole table (naming the whole table is the caption's job).

Never use a table for page layout

Before CSS, whole pages were built inside tables: a row for the banner, cells for the menu and articles. Today that is wrong. Layout tables hurt accessibility and block responsiveness; tables are meant for tabular data only.

- **Accessibility:** a screen reader may announce "table, 2 columns" and read out row and column positions that mean nothing (browsers try to guess layout tables, but the guess is unreliable).
- **Responsiveness:** a table's grid does not reflow into one column on a phone unless extra CSS tears the table apart; left alone, the cells stay side by side, squeezed or scrolling sideways.
- **Meaning:** the markup claims the page is data when it is not.

The modern way is semantic regions (`<header>`, `<nav>`, `<main>`, `<aside>`, `<footer>`) arranged with CSS Flexbox or Grid.

Trap: the reason is *not* that tables cannot hold images (a cell can hold images, links, lists, even a form), *not* that browsers dropped `<table>` (it is a current, fully supported element and the right choice for data), and *not* speed (tables do not "always" load slower than divs, and speed is not the objection). The real problems are accessibility and reflow.

If you inherit a layout table you cannot rewrite (old HTML emails are the classic case), `<table role="presentation">` tells assistive technology to ignore its data meaning. It changes nothing on screen: it does not hide borders, make the table sortable or make the cells stack on a phone.

CHECK YOURSELF

10.1 A mess-menu table has four columns: Day | Breakfast | Lunch | Dinner. On the Sunday row, the words 'Special brunch' must fill the Breakfast and Lunch slots as one cell. Which row markup keeps the grid correct?

a. <tr><th>Sunday</th><td rowspan="2">Special brunch</td><td>Pulao</td></tr>

b. <tr><th>Sunday</th><td colspan="2">Special brunch</td><td></td><td>Pulao</td></tr>

c. <tr><th>Sunday</th><td colspan="2">Special brunch</td><td>Pulao</td></tr>

d. <tr><th>Sunday</th><td width="2">Special brunch</td><td>Pulao</td></tr>

10.2 A library-hours table has columns Day | Shift | Hours. Saturday has two shifts written as two consecutive rows, and the word 'Saturday' must appear only once, covering both rows. What should the markup do?

a. Give the Saturday cell rowspan="2" and start the second Saturday row with an empty <td> for the Day column
b. Give the Saturday cell rowspan="2" and leave the Day cell out of the second Saturday row
c. Place the two Saturday rows in their own <tbody>, because a tbody automatically shares its first cell
d. Give the Saturday cell colspan="2" so that it reaches down into the row below

10.3 A hostel laundry rate card already starts with a <caption> and uses <th> for every heading. Two of its rows are <tr><td></td><th>Wash</th><th>Wash + iron</th></tr> and <tr><th>Shirt</th><td>15</td><td>25</td></tr>. Which pair of additions tells a screen reader outright that 25 sits under 'Wash + iron' and beside 'Shirt'?

a. role="presentation" on the <table>, and a title on the Shirt heading
b. A <tfoot> row repeating both headings, and a larger font for Shirt
c. colspan="2" on the Shirt heading, and a border round each price
d. scope="col" on the two column headings, and scope="row" on Shirt

10.4 An old newsletter page is one big <table>: the banner in the top row, the menu in the left cell, articles in the right cell. On a phone the menu and articles stay squeezed side by side, and a screen reader reads out table positions. What is the right fix?

a. Rebuild it with <header>, <nav> and <main> regions, arranged by CSS Flexbox or Grid instead of cells
b. Keep the table but add role="presentation", which also makes its cells stack into one column on phones
c. Hide the table borders with CSS, so phones and screen readers treat it as an ordinary page
d. Fix the table at width="980", so a phone has room to show the menu and articles side by side

11 Responsive content structuring

Textbook chapter 21

What responsive means

A **responsive** page is **one** HTML document that adapts to whatever screen shows it, from a 360-pixel phone to a wide monitor: text reflows, columns stack, images shrink. There is no separate mobile site. Think of water: it takes the shape of any glass you pour it into. Three things work together:

1. HTML with a clean, logical structure (this section);
2. CSS media queries that rearrange it for each screen width (Unit 3);
3. images and videos that can scale.

The viewport meta tag

Mobile browsers were built when the Web was all desktop pages. So, by default, a phone **pretends** its screen is about 980 pixels wide, lays the page out at that width, then zooms out to squeeze it onto the small screen. The result is a miniature desktop page whose text is too small to read without pinching. One line in the `<head>` stops the pretending:

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

- `width=device-width`: lay the page out at the device's real screen width.
- `initial-scale=1.0`: open at 100% zoom, not zoomed out.

Without it, CSS media queries also measure the fake 980-pixel width, so phone-only styles never switch on. That is why the viewport tag is the one meta tag that phones need to show a page at its real size.

Meta tag	What it does	Makes the page fit a phone?
`<meta charset="UTF-8">`	Sets the character encoding, so the rupee sign ₹ and Hindi letters display correctly	No (but still needed on every page)
`<meta name="description" content="...">`	A summary that search engines may show under the link	No
`<meta name="author" content="...">`	Records who wrote the page	No

`<meta name="viewport" content="...">`	Sets the layout width and starting zoom on mobile	**Yes**

> **Trap:** a page with no viewport tag still loads normally, and nothing on it is hidden. The browser does not swap in some built-in "mobile stylesheet" either; it simply shows the desktop layout shrunk down. Also, never add `user-scalable=no` or `maximum-scale=1`: blocking zoom hurts people with weak eyesight.

Mobile-first and logical source order

Mobile-first means you start with the smallest screen, then add improvements as screens get bigger: first a single column that works on a phone; when there is more room, CSS adds a second column, then a third. Working the other way (desktop-first: shrinking a big design down) gives cramped, compromised phone pages. Mobile-first also forces good priorities: only the important content fits a small screen, so you decide what matters first. It does not mean "desktop only", and it has nothing to do with print layouts.

The single-column phone view is simply the HTML in the order it was written. So **logical source order** is the foundation: header, navigation, main content, related content, footer. Get the order right and the page already works on a phone before any CSS exists; screen readers and search engines read it in that order too.

```
<header>Tech fest</header>
<nav>Events | Register</nav>
<div class="cols">
  <main>Schedule and venues</main>
  <aside>Sponsors</aside>
</div>
<footer>Organising team</footer>
```

On a phone these simply stack. Later, one CSS rule puts main and aside side by side only when there is room, without touching the HTML:

```
@media (min-width: 700px) {
  .cols { display: flex; }
}
```

Flexible, scalable media

A 1600-pixel photo shown at its full 1600 pixels overflows a 390-pixel phone and drags the whole page sideways. In a responsive page, images and videos must be able to shrink and grow with the box that holds them; they are never locked to a pixel width wider than a small screen. The standard CSS rule:

```
img, video { max-width: 100%; height: auto; }
```

- `max-width: 100%`: never wider than the box it sits in.
- `height: auto`: the height follows, so the picture keeps its shape instead of stretching.

Keep writing `width` and `height` attributes on `<img>`: they tell the browser the image's proportions so the page does not jump while it loads, and the CSS above still lets it shrink.

> **Trap:** the wrong habit is *fixing* the displayed size in pixels so the image cannot shrink. Hiding images on phones, swapping them for text, or sending them only to desktops removes content; responsive design keeps the content and resizes it.

Shrinking a huge file still downloads the huge file. `srcset` lists several sizes, `sizes` says how wide the image will appear, and the browser picks the smallest file that is still sharp. `<picture>` can even give phones a differently cropped image.

```
<img src="gate-800.jpg" alt="College main gate" width="800" height="450"
     srcset="gate-400.jpg 400w, gate-800.jpg 800w, gate-1600.jpg 1600w"
     sizes="(max-width: 600px) 100vw, 800px">
```

A wide data table cannot become one column without losing its meaning, so let it scroll inside its own box while the page stays still: wrap it in `<div class="scroll-x">` and give that class `overflow-x: auto` in CSS.

Habits that help and habits that hurt

Habit that hurts	Why it breaks on a phone	Do this instead
Fixed pixel widths on containers (a CSS `width: 900px`, or an old `width="900"` attribute on a table)	Wider than the screen, so the page scrolls sideways	Fluid widths: percentages or `max-width`
Fixed pixel heights on sections	Text wraps onto more lines on a narrow screen and spills out	Let the content decide the height
Layout tables, nested inside each other	A table grid cannot reflow into one column	Semantic regions + Flexbox or Grid
Images with no `max-width`	They overflow their container	`max-width: 100%`, `srcset`, `<picture>`
No viewport tag	Shrunken desktop page	Put it in every page's head
Tiny tap targets; wrong input types	Hard to tap; wrong keyboard	Large buttons; `type="email"`, `type="tel"`

> Before any CSS is written, the best preparation is putting content in logical source order and making images and videos flexible, plus the viewport tag. Then the CSS only has to add columns when space allows.

Testing: in Chrome press F12, then Ctrl+Shift+M for device mode, and drag the width; then try a real phone, because emulators do not reproduce touch or slow networks. A page passes when no sideways scrollbar appears at any width, text is readable without zooming, images fit and every control can be tapped.

CHECK YOURSELF

11.1 A portfolio site looks fine on a laptop. On a phone it appears as a miniature of the desktop page, visitors must pinch-zoom to read, and the CSS rule @media (max-width: 600px) never seems to apply. What is the most likely cause?

a. The <title> text is longer than 60 characters
b. The page has no <meta name="keywords"> tag, so the phone cannot tell it is meant for mobile
c. The <head> is missing <meta name="viewport" content="width=device-width, initial-scale=1">, so the phone lays the page out about 980px wide and zooms out
d. The file is saved as index.htm instead of index.html

11.2 A team is starting a new canteen-ordering website. Which plan is mobile-first and keeps the HTML ready for a responsive layout?

a. Design the 1366px laptop layout first, then hide sections one by one until it fits a phone
b. Build a separate mobile site with fixed 360px widths and a second desktop site with fixed 1200px widths
c. Fix every section at height: 500px and every photo at width: 1200px so nothing moves between devices
d. Write the content in a logical order (header, nav, main, footer) as one column for a 360px phone, let photos shrink with max-width: 100%, then add columns with min-width media queries as screens get wider

11.3 A 1600-pixel-wide photo of the cricket ground makes a phone page scroll sideways. Which fix keeps the photo on the page and makes it fit the screen?

a. Add overflow: hidden to the <body> so the sideways scrollbar disappears
b. Add img { max-width: 100%; height: auto; } so the photo shrinks to its container and keeps its shape
c. Change width="1600" to width="1600px" so the browser measures it exactly
d. Wrap the photo in a <figure> element, which phones shrink automatically

A Answers to "Check yourself"

No.	Answer	Why
1.1	(b) The student used the Internet but not the Web, since none of it fetched HTML pages over HTTP	The Internet carries all these services while the Web is only the HTML-pages-over-HTTP service; having an IP address puts a device on the Internet, not on the Web.
1.2	(c) Looking up the recipe in its database and returning it as JSON	A server stores data, runs logic and responds with content such as JSON or HTML; turning a name into an IP address is the separate DNS step, and history stays on the phone.
1.3	(d) The browser uses port 443 and data in transit is TLS-encrypted	HTTPS is the same HTTP conversation wrapped in TLS on default port 443; the path, the HTML and the content type all stay exactly as they were.
1.4	(a) The text line GET /notice.html HTTP/1.1 sent from a browser to a server	HTTP is the request-response protocol whose HTTP/1.1 messages look like that GET line; the radio signal is physical infrastructure that carries HTTP traffic but is not the protocol itself.
1.5	(c) T, R, P, S, Q	The cycle is user action, request, server processing, response, render; T, S, R, P, Q is tempting but has the response arriving before any request was sent.
1.6	(b) Path: /fest/2026/schedule.html	The path runs from the first / after the domain up to the ?, so it is /fest/2026/schedule.html; /fest/2026 is only part of the path, while the query string is ?day=2&hall=B.
1.7	(a) The domain is admissions.greenvalley.ac.in and the protocol is https	The domain is the whole host name between :// and the first /, and https before :// is the protocol; 'in' is only the top-level domain and /apply is the path.
2.1	(c) <!doctype html>	The doctype <!doctype html> (case does not matter) switches on standards mode; HTML5 has no version attribute, meta tag or head type that does this job.
2.2	(b) <table> listing the campus bus timings	A table is visible content, so it goes in the body; the author meta tag, the favicon link and a style block are information about the page and are correct in the head.
2.3	(a) <meta charset="UTF-8">	charset declares the character encoding so the bytes become the right characters; the viewport tag only controls layout width on phones, and a stylesheet can only restyle characters that were already decoded.
2.4	(d) The content of <meta name="description">	The meta description supplies the summary under the headline in search results; the <title> supplies the clickable headline itself and the tab text, while the <h1> is the visible heading on the page.
2.5	(c) src, alt and width are attributes of the img element	Each name="value" pair in an opening tag is an attribute that configures the element; a CSS declaration would look like width: 300px; and img is a void element with no content and no closing tag.
2.6	(b) <section>	<section> is block-level like div and p; , <a> and <code> are inline, so they flow inside a line of text and take only the width of their content.
2.7	(d) Safety Rules <img src="goggles.png" alt="Pair of clear safety goggles">	Only the last version closes the heading with a matching </h3> and gives the image alt text while keeping its src; the first adds the alt but closes with </h4>, which does not match the opening <h3>.

No.	Answer	Why
3.1	(b) , <i>,	gives the warning real importance, <i> marks a foreign term without stress and marks the stressed word; <b> would only make the warning look bold without telling a screen reader that it matters.
3.2	(a) Change the h4 to an h3	A sub-topic under an h2 must be an h3 so no level is skipped; adding a second h1 or removing the only h1 would leave the page without one clear main topic.
3.3	(d) , , <dl>	Unordered snacks take , a sequence whose order matters takes , and name-plus-description pairs take <dl> with <dt>/<dd>; putting the ordering steps in a <dl> would confuse a sequence with term-and-definition pairs.
3.4	(c) <ol type="I"> - Unit one - HTML - CSS	type="I" gives I, II, III and the sub-list sits inside the first unit's ; the first option has the right type but puts the directly inside the , which is invalid.
4.1	(c) Refunds	A hash followed by the id is a fragment link to that element on the current page; href="refunds.html" would instead load a separate file with that name.
4.2	(b) The shop tab taking control of the portfolio tab	noopener stops the new tab from reaching back to control the tab that opened it (noreferrer additionally hides which page sent the visitor); the value is a safety measure and has nothing to do with loading speed, search listings or downloads.
4.3	(d) href="tel:+911800123456" and href="mailto:office@hostel.example"	tel: starts a call and mailto: opens a new email; sms: is a real scheme but opens a text message rather than a call, and telephone: and inbox: are not recognised at all.
4.4	(a) Photo: alt="Captain lifting the inter-college football trophy"; line: alt=""; icon: alt="Search"	The photo is described, decoration gets an empty but present alt and a functional icon is named by its action; leaving alt off the line can make a screen reader read its file name, and "Magnifying glass icon" describes the look, not what the button does.
4.5	(b) srcset	srcset lists the candidate files with their widths for the browser to choose from; sizes only tells the browser how wide the image will be displayed.
5.1	(c) <circle cx="30" cy="45" r="12" />	A circle is placed by its centre (cx, cy) and sized by r; x1/y1 belong to a line and radius is not an SVG attribute, so those versions are ignored.
5.2	(d) <line x1="10" y1="60" x2="190" y2="60" stroke="gray" />	A line needs x1, y1, x2, y2 for its endpoints and a stroke to be seen; the fill version has correct endpoints but stays invisible because a line has no inside to fill.
5.3	(b) 160 px	The viewBox grid of 50 units is drawn across 200 px (4 px per unit), so the 40-unit diameter becomes 160 px; 40 px forgets that the viewBox maps its units onto the rendered size.
5.4	(a) The polyline shows two joined segments with an open end; the polygon adds a third side back to (10,10)	Only polygon closes the shape back to its first point; neither element draws curves, so the smooth-bend option is wrong.
5.5	(c) A closed triangle with corners at (20,20), (60,20) and (60,60)	h 40 and v 40 move relative to the pen, giving (60,20) then (60,60), and z adds the straight closing side back to the M point (20,20); the open-corner option ignores what z does.
5.6	(b) fill="none" stroke="orange" stroke-dasharray="5,3"	Hollow means fill none, the outline colour is stroke, and dash-then-gap goes in stroke-dasharray; stroke-width sets thickness, not a dash pattern.
6.1	(c) <video src="intro.mp4" controls poster="lecturer.png"></video>	controls draws the player buttons and poster sets the pre-play image; the preload version has the right image but no buttons, because preload only hints at downloading.

No.	Answer	Why
6.2	(a) It plays gym.webm and restarts it from the beginning each time it finishes	The browser takes the first source in the list it can play and ignores the rest, and loop restarts that clip; sources are a fallback shortlist, not a playlist.
6.3	(d) Add title="Live weather for the campus"	title gives the frame its accessible name, which the screen reader announces; alt is not an iframe attribute, so it does nothing here.
7.1	(b) She can jump straight to the menu or the film listings using landmark shortcuts	nav and main are exposed as landmarks a screen reader can jump between, while a div page is still indexed by search engines (just with less structural information), so the search option overstates the difference.
7.2	(d) , because this wrapper exists only for the layout and carries no meaning	A layout-only wrapper carries no meaning, which is exactly the job of div; a second visible main is invalid because a page may never have more than one visible main element.
7.3	(a) One <nav aria-label="Result pages"> around a single list holding all four links	Pagination is one group of navigation links, so it is wrapped once in a single nav; wrapping each link separately creates many meaningless landmarks.
7.4	(c) P = article, Q = section, R = aside	The report stands alone (article), the headed squad block is a thematic part of this page (section) and the related-stories box is useful but removable (aside); the first option swaps the meanings of section and article.
7.5	(b) <header> holds the logo and menu; <main> holds the room cards and a "Nearby sights" <aside>; <footer> holds the phone and copyright	Introductory content belongs in header, the page's own content (including a related aside) in main, and closing contact details in footer; putting the logo and menu in an aside is tempting because they repeat, but they are introduction and navigation, not related extras.
7.6	(c) Screen-reader users lose the labels they use to identify and jump between the sections	A section's heading is what assistive technology uses to name, summarise and navigate it; the page still renders and is still indexed without it, so the rendering and indexing options are false.
8.1	(d) W Perceivable, X Operable, Y Understandable, Z Robust	Missing captions block perceiving sound, a mouse-only picker cannot be operated by keyboard, a vague error cannot be understood, and a control that fails in other browsers and assistive technology is not robust; the second option wrongly treats the mouse-only picker as a Robust failure.
8.2	(b) Every Level A and every Level AA success criterion of the Web Content Accessibility Guidelines	WCAG (Web Content Accessibility Guidelines) levels are cumulative, so AA means all A plus all AA criteria and nothing from AAA is required; the first option is wrong because AA builds on Level A rather than replacing it.
8.3	(b) <button aria-label="Mute"><svg aria-hidden="true">…</svg></button> and <button>Share <svg aria-hidden="true">…</svg></button>	aria-label names the icon-only Mute button, the visible word Share names the second button, and aria-hidden goes only on the decorative icons; the third option hides the word Share itself, leaving that button with no name.
8.4	(a) Each thumbnail can now be reached with Tab, in the same order as it appears in the source	tabindex="0" only adds the element to the normal focus order at its source position; it gives no role and no Enter or Space behaviour, which still need a script.
8.5	(c) <button type="submit">Place order</button>, since it is focusable and submits on Enter or Space with no extra code	A native <button> already has the button role, keyboard focus and form submission built in; role="button" on a span changes only what is announced, not how it behaves, so Enter and Space still do nothing.
9.1	(c) They are sent to /hostel/apply inside the body of the request	action names the destination URL and post packs the name=value pairs into the request body; a ? query string in the URL is what get does, and post encrypts nothing.

No.	Answer	Why
9.2	(b) The hint vanishes once typing starts and may not be announced	A placeholder is only a temporary hint that disappears on typing and may be skipped by screen readers; it has no effect on submission, so the claim that browsers refuse to submit is wrong.
9.3	(d) All three can be selected at the same time	Radios form one exclusive group only when they share a name; with three different names each is a group of one, so choosing one never clears the others.
9.4	(a) The passphrase in plain, readable characters	There is no type "secure"; browsers turn any unrecognised type into type="text" without complaint, and only type="password" masks characters as dots.
9.5	(d) <button>Save</button> <button type="button">Preview</button>	A <button> with no type inside a form defaults to submit, so Save sends the form, and type="button" gives Preview no default action; leaving both untyped is tempting but makes Preview submit the post too.
9.6	(b) The form is sent, because minlength does not judge an empty box	An empty value never counts as too short, so without required it passes; once something is typed, minlength="6" demands at least six characters.
9.7	(c) Submission is stopped, the box is focused, and the error message includes the title's text about the expected format	b214 fails the pattern (lower-case letter, no hyphen), so the browser blocks the form and shows its mismatch message with the title as the format hint; the browser never fixes the value, and the server's key comes from name, not title.
9.8	(a) A check in the server code that validates the quantity again	Anything in the browser can be edited or skipped, so only server-side validation, which the user cannot reach, enforces the limit; switching to post changes where the data travels, not what is allowed.
9.9	(c) office@	The standard needs a domain after the @, so office@ fails; office@hostel is tempting but valid, because a dot after the @ is allowed but not required.
10.1	(c) <tr><th>Sunday</th><td colspan="2">Special brunch</td><td>Pulao</td></tr>	colspan="2" widens the cell sideways over two columns, so Sunday + 2 + Pulao adds up to four; rowspan would stretch it downwards into the next row instead and leave this row one column short.
10.2	(b) Give the Saturday cell rowspan="2" and leave the Day cell out of the second Saturday row	rowspan="2" stretches the label down over both rows, and since it already fills the Day slot of the second row that row must omit the cell; adding an empty <td> pushes the row one column too wide.
10.3	(d) scope="col" on the two column headings, and scope="row" on Shirt	scope="col" and scope="row" declare which column and which row each heading labels, so 25 is announced with both headings; role="presentation" would do the reverse and switch off the table's data meaning altogether, while a title is only a tooltip, a repeated tfoot row and colspan link no heading to any value, and fonts and borders change only the looks.
10.4	(a) Rebuild it with <header>, <nav> and <main> regions, arranged by CSS Flexbox or Grid instead of cells	Semantic regions laid out with Flexbox or Grid can reflow into one column and are not announced as data; role="presentation" only switches off the table meaning for assistive technology and does nothing to make the cells stack.
11.1	(c) The <head> is missing <meta name="viewport" content="width=device-width, initial-scale=1">, so the phone lays the page out about 980px wide and zooms out	Without the viewport tag the phone pretends to be a roughly 980px desktop, so max-width: 600px is never true and the shrunken layout appears; a keywords tag has nothing to do with layout width.
11.2	(d) Write the content in a logical order (header, nav, main, footer) as one column for a 360px phone, let photos shrink with max-width: 100%, then add columns with min-width media queries as screens get wider	Mobile-first starts from the smallest screen and adds enhancements upward on top of a logical source order and scalable media; starting from the laptop design and cutting it down is desktop-first, the opposite direction.

No.	Answer	Why
11.3	(b) Add img { max-width: 100%; height: auto; } so the photo shrinks to its container and keeps its shape	max-width: 100% lets the image scale down with its container while height: auto keeps its proportions; hiding the overflow on body only removes the scrollbar and crops the photo instead of fitting it.