1. Intro to the Web — How web communication is established

Core idea (one-sentence)

A user's **browser (client)** asks a **server** for a resource (page, image, data) using a protocol (HTTP/HTTPS). The server answers with a response (status code + headers + body).

Steps of a simple web request

1. **User types/requests a URL** or clicks a link.
2. **DNS lookup**: Browser asks DNS to resolve domain → IP address.
3. **TCP connection**: Browser opens a TCP connection (often via TCP 3-way handshake) to the server IP and port (default 80 for HTTP, 443 for HTTPS).
4. **TLS handshake** (HTTPS only): Client and server negotiate encryption keys.
5. **HTTP request**: Browser sends a request (method + URI + headers [+ body]).
6. **Server processes** request (maybe calls application logic, DB).
7. **HTTP response**: Server sends status code, headers, and body (HTML/CSS/JSON).
8. **Browser renders** the response, fetches subresources (images, CSS, JS).
9. **Connection close or reuse** (HTTP/1.1 keep-alive / HTTP/2 multiplexing).

Protocols: HTTP vs HTTPS

- **HTTP (HyperText Transfer Protocol)** — plain, unencrypted text; stateless request/response.
- **HTTPS** — HTTP over TLS/SSL: **encrypted**, protects confidentiality and integrity.
  Key difference: HTTPS encrypts the transport (the "locked box" analogy). Use HTTPS for login, payments, forms.

HTTP Request — basic parts

- **Request line:** METHOD URI HTTP/VERSION (e.g., GET /index.html HTTP/1.1)
- **Common methods:**
  - GET — fetch resource, safe/idempotent (no body).
  - POST — submit data (forms), can create resources.
  - PUT — replace/update resource.
  - DELETE — delete resource.
  - PATCH — partial update.
- **Headers:** Host, Content-Type, Accept, User-Agent, Cookie, etc.

- **Body:** optional (for POST/PUT with payload).

HTTP Response — basic parts

- **Status line:** HTTP/1.1 200 OK
- **Status code classes:**
    - 1xx informational, 2xx success (200 OK), 3xx redirect (301), 4xx client error (404 Not Found, 422 Unprocessable Entity), 5xx server error (500).
- **Headers:** Content-Type, Content-Length, Set-Cookie, Cache-Control, etc.
- **Body:** HTML/CSS/JSON/image.

Tip: 4xx = client must change request (e.g., validation); 5xx = server side problem.

Server-Client Model

- **Client (browser)**: initiates requests, renders responses, handles UI, stores small client state (cookies, localStorage).
- **Server**: listens for requests, processes, communicates with databases, returns responses.
- **Statelessness:** HTTP treats each request independently — the server does not remember previous requests unless state is stored explicitly (cookies, sessions, tokens).

How web communication handles mixed content & blocking

- **Mixed content**: If page served over HTTPS, fetching a subresource over HTTP may be blocked by the browser (insecure → blocked). That's why http://cdn.example.com/banner.jpg can be blocked when main page is https://....

---

2. Servers: common types & roles

Web Server

- Serves static files (HTML, CSS, images) and forwards dynamic requests to application servers (Apache, Nginx, IIS).

Application Server

- Runs business logic and dynamic content (Node.js, Tomcat, Django app). Processes requests and interacts with DB.

Database Server

- Stores persistent data (MySQL, PostgreSQL, MongoDB). Application layer queries DB.

Proxy Server

- Intermediary between client and server (caching, load balancing, security). Examples: reverse proxy, forward proxy.

DNS Server

- Resolves domain names to IP addresses.

DHCP Server

- Assigns IP addresses in a network.

Auth (Authentication) Server

- Handles user login, tokens (OAuth, JWT), session management.

Game Server

- Real-time session handling for games; high emphasis on low latency.

FTP / SMTP

- **FTP (File Transfer Protocol)** — for file transfers (legacy).
- **SMTP (Simple Mail Transfer Protocol)** — for sending email.

---

3. Architectures: Client-Server, 3-Tier, Peer-to-Peer

Client-Server (2-tier)

- Client requests → Server responds. Simple, many web apps start here.

Three-Tier Architecture

- **Presentation (Client)** — UI: HTML/CSS/JS.
- **Application (Logic)** — Server side code; business rules.
- **Data (Database)** — persistence layer.
  Benefit: separation of concerns, easier scaling & maintenance.

Peer-to-Peer (P2P)

- No central server; peers connect directly and share resources (files, streams). Each node acts as client & server. Used in file sharing, WebRTC sometimes uses P2P for media.

---

4. Request Methods & Common Status Codes (cheat list)

Methods: GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD.
Status codes: 200 OK, 201 Created, 301 Moved Permanently, 302 Found, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 422 Unprocessable Entity (validation error), 500 Internal Server Error.

---

5. Intro to HTML — structure, tags, semantics

Basic HTML Document

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Title</title>
  </head>
  <body>
    <header></header>
    <main></main>
    <footer></footer>
  </body>
</html>
```

Common tags & their purpose

- **<head>**: metadata, <title>, links to CSS/JS, <meta> tags (charset, viewport, description, Open Graph). *Not visible content; used by browser and social sharing.*
- **<body>**: visible content.
- **Text & structure:** <h1>–<h6>, <p>, , , <code>.
- **Grouping:** (generic block), (generic inline).
- **Navigation & layout:** <header>, <nav>, <main>, <section>, <article>, <aside>, <footer> — semantic tags that give meaning and help accessibility/SEO.
- **Links & images:** <a href="...">, <img src="..." alt="...">. **Always include alt** for accessibility and for cases when image fails.
- **Forms & inputs:** <form action="..." method="POST"> with input controls.
- **Lists & tables:** , <ol>, <li>; <table>, <caption>, <thead>, <tbody>, <tfoot>, <tr>, <th>, <td>.
- **Field grouping:** <fieldset> groups related form fields; <legend> gives a caption.

Important HTML attributes

- **id** — unique identifier, used for CSS/JS.
- **class** — can be reused for styling/multiple elements.

- **name** — form control name used in form submission.
- **value** — default value for inputs.
- **required** — marks field required for form submission.
- **src / href** — resource URLs for images and links.
- **alt** — alternative text for images (accessibility + fallback).
- **type** — input type: text, password, number, email, file, checkbox, radio, submit, reset, hidden.
- **enctype="multipart/form-data"** — needed for file uploads in forms.

Meta tags commonly used

- <meta charset="utf-8"> — encoding.
- <meta name="viewport" content="width=device-width, initial-scale=1"> — mobile scaling.
- <meta name="description" content="..."> — SEO snippet.
- Open Graph / Twitter cards for social sharing: <meta property="og:title" content="...">, <meta property="og:image" content="...">.

---

6. Semantic HTML — why it matters

- Semantic tags describe **meaning** (easier for screen readers, SEO, maintainability).
- e.g., <header>, <nav>, <main>, <article>, <section>, <footer>, <aside>.

---

7. Forms & Tables — important notes

Forms

- action — URL where form sent.
- method — GET (data in URL) or POST (data in body). **Default is GET** if omitted.
- Use proper type for built-in validation (type="email" triggers email format validation).
- For file upload use enctype="multipart/form-data" and <input type="file">.
- Use required, min, max, pattern for constraint validation. Client validation can be bypassed — always validate server side as well.

Tables

- Use <caption> for table title.
- Use <th> for header cells (screen readers treat them specially).
- Merge cells: colspan (columns), rowspan (rows).

---

8. Intro to CSS — purpose & inclusion types

What is CSS?

- CSS styles the HTML—colors, layout, spacing, fonts, responsive behavior.

Ways to include CSS

1. **External CSS** — <link rel="stylesheet" href="styles.css"> (best for reuse).
2. **Internal CSS** — <style> block in <head> (page specific).
3. **Inline CSS** — style="..." in tags (overrides others, not recommended).

Best practice: External CSS for site-wide styles.

CSS Units

- **Absolute units:** px, pt, cm, mm, in — fixed physical sizes (less responsive).
- **Relative units:** %, em, rem, vw, vh — scale with parent/font/viewport (better for responsiveness).
    - em: relative to font size of the element.
    - rem: relative to root font size.
    - vw, vh: 1% of viewport width/height.

CSS Selectors (covering common types)

- **Universal:** * { } — selects everything.
- **Element (type) selector:** p { } — selects all <p>.
- **Class selector:** .btn { } — selects elements with class="btn".
- **ID selector:** #logo { } — selects element with id="logo" (unique).
- **Attribute selector:** input[type="text"] { }
- **Group selector:** h1, h2, h3 { }
- **Descendant selector:** .nav a { } (any a inside .nav).
- **Child selector:** ul > li { } (direct children only).
- **Pseudo-classes:** a:hover { }, input:focus { }, li:first-child { }.
- **Pseudo-elements:** p::first-line { }, p::before { content: "• "; }.

Specificity order (low → high): element < class < id < inline styles. Use !important sparingly.

Box Model (core concept)

Every element is a box:

[margin] [border] [padding] [content]

- **Content**: width/height set by you.
- **Padding**: space between content and border (adds to size).
- **Border**: border thickness.
- **Margin**: space outside border (separates elements).

Total width = width + padding-left + padding-right + border-left + border-right + margin-left + margin-right (unless box-sizing: border-box).

Tip: box-sizing: border-box makes width include padding+border (easier layouts).

Display property (detailed)

- display: block; — element starts on new line, takes full width.
- display: inline; — sits with text, cannot set width/height.
- display: inline-block; — inline layout, but can set width/height (great for horizontal menus).
- display: flex; — flex container for flexible 1D layouts (rows/columns).
- display: grid; — grid container for 2D layouts (rows and columns).
- display: none; — removes element from page flow (no space taken).

Use cases: block for sections, inline for text flows, inline-block for horizontally aligned boxes without floats.

Positioning (detailed)

- static — default, flows naturally.
- relative — offset from normal position via top/left but still takes original space.
- absolute — removed from document flow and positioned relative to nearest positioned ancestor (non-static).
- fixed — fixed relative to viewport (stays during scroll).
- sticky — behaves like relative until crossing threshold, then sticks like fixed.

z-index: controls stacking order; only works on positioned elements (relative, absolute, fixed, sticky). Higher z-index = on top.

Overflow

- overflow: visible; (default) → content can overflow box.
- overflow: hidden; → overflow is clipped.
- overflow: auto; → add scrollbars if needed.
- overflow: scroll; → always show scrollbars.

Outline

- outline draws a line outside the element, does **not** take layout space. Useful for debugging or accessibility focus states.

---

9. CSS Practical Tips & Examples

Horizontal menu with inline-block

<nav>

  Home

  About

</nav>

<style>

.menu { display: inline-block; padding: 10px 16px; }

</style>

Sticky header

header { position: sticky; top: 0; background: white; z-index: 1000; }

Centering with flex

.container { display: flex; justify-content: center; align-items: center; }

---

10. Basic JavaScript — essentials

What is JavaScript?

- Scripting language that runs in browsers (and servers via Node.js). Adds interactivity, dynamic updates, and client-side logic.

Variables & scope

- var — function-scoped (legacy).
- let — block-scoped.
- const — block-scoped, cannot reassign.

Types

- Primitives: string, number, boolean, null, undefined, symbol.
- Complex: object, array, function.

Operations

- Arithmetic: +, -, *, /, %, exponent **.
- Comparison: == (loose), === (strict), !=, !==, >, <, >=, <=.
- Logical: &&, ||, !.

Tip: Prefer === to avoid type coercion surprises.

Type Conversion

- **Implicit** coercion: JS may convert types automatically ("5" + 1 → "51").
- **Explicit** conversion: Number("10"), String(100), Boolean(0); parseInt("20 mins") → 20 (stops at first non-digit). Use parseFloat for decimals.

Flow Control

- if/else, ternary condition ? a : b, switch (value) { case ... }.

Loops

- for (let i=0;i<n;i++) {}
- while(condition) {}
- do { } while(condition)
- for...in — iterate object keys
- for...of — iterate iterable values (arrays, strings)

Functions

- **Declaration**:

function add(a,b){ return a+b; }

- **Expression / Arrow**:

const add = (a,b) => a + b;

- **Scope note:** Arrow functions don't have their own this.

Strings — common methods

- length, slice(start,end), substring(start,end), toUpperCase(), toLowerCase(), replace(old,new), trim(), split(delim).

Arrays — common operations

- push, pop, shift, unshift, indexOf, includes, map, filter, reduce, sort, reverse.
- typeof array returns "object" — arrays are objects in JS. Use Array.isArray(arr) to check arrays.

---

11. DOM Manipulation with JavaScript (practical)

Access elements

```
document.getElementById('id');

document.getElementsByClassName('cls'); // HTMLCollection

document.querySelector('.cls'); // first match

document.querySelectorAll('li'); // NodeList (static)
```

Insert element

```
const p = document.createElement('p');

p.textContent = "Hello";

document.body.appendChild(p);          // add to end

const parent = document.getElementById('list');

parent.insertBefore(p, parent.firstChild); // insert at start
```

Update content

```
const el = document.getElementById('title');

el.textContent = 'New title';

el.innerHTML = '<strong>Bold</strong>'; // use carefully (XSS risk)
```

Delete element

```
const el = document.getElementById('old');

el.remove(); // modern

// or el.parentNode.removeChild(el);
```

Style manipulation

```
el.style.backgroundColor = 'yellow';

el.classList.add('highlight'); // preferred — toggle classes, keep CSS in stylesheet
```

```
el.classList.remove('highlight');
```

Form validation (simple example)

```
function validateForm(e) {
  const name = document.getElementById('name').value.trim();
  if (!/^[A-Za-z\s]+$/.test(name)) {
    alert('Name must contain only letters');
    e.preventDefault(); // stop submission
  }
}
document.querySelector('form').addEventListener('submit', validateForm);
```

---

12. Accessibility, SEO & Best Practices

- Always use **semantic HTML** (<header>, <main>, <nav>) and **alt** attributes for images.
- Use label elements for form inputs: <label for="email">Email</label><input id="email">.
- Keep CSS in external files, minimize inline styles.
- Validate input both client & server side.
- Use HTTPS site-wide.
- Use rel="noopener noreferrer" when using target="_blank" to prevent opener vulnerabilities.

---

13. Quick Practical Cheat Sheet (Common MCQ Answers & Rationale)

- **NAT** — allows multiple devices to share one public IP (router feature).
- **<head>** — store metadata and links to stylesheets/scripts.
- **target="_blank"** — opens links in a new tab.
- **Attribute** — property on an HTML element (name="value").
- **JavaScript inventor** — Brendan Eich (1995), standardized by ECMA (1997).
- **Web server role** — serve static & dynamic web pages.
- **URI example https://example.com:7000/...** — port is 7000.
- **Merge columns** — colspan.
- **Space inside border** — padding.
- **Selector for only text inputs** — input[type="text"].

- **Add space between elements without changing internal layout** — margin.
- **parseInt("20 mins") → 20** (parseInt stops at first non-digit).
- **Block scoped variables** — let, const.
- **Create new post** — use POST.
- **HTTP statelessness** — server does not remember previous requests, so HTTP is **stateless**.
- **Array declaration** — var arr = [] or let arr = ["a","b"].
- **typeof array** returns "object" (use Array.isArray() to detect arrays).
- **Convert Date to string** — d.toISOString() or d.toUTCString(); (avoid non-standard parseString).
- **Select elements by class as static NodeList** — document.querySelectorAll('.item').
- **Site-wide styles best via External CSS** — <link rel="stylesheet" href="style.css">.
- **Add click handler** — document.getElementById('myBtn').addEventListener('click', myFunction);
- **Selector for ID header** — #main-header.
- **Hoisting & Temporal Dead Zone** — accessing let/const before declaration throws ReferenceError.
- **display:block** — element takes full width and starts on new line.
- **position: sticky** — element sticks to top after scrolling past it.
- **display:none** removes element from layout normal flow (causes reflow).
- **Form default method** — GET if missing.
- **onsubmit** attribute used to call validation on form submit.
- **$_FILES (PHP reference)** — handles uploaded file metadata (if you are comparing server side).
  (These match many MCQs you provided — keep these quick facts memorized.)

---

14. Short Exam-Style Examples & Explanations (common asked concepts)

Example: Why use alt in <img>?

- **Answer:** Accessibility + fallback. If image fails to load, alt shows screen readers' description and browsers may display that text.

Example: Why POST vs GET?

- GET puts parameters in URL (bookmarkable, visible, length limits), POST sends data in body (better for sensitive info and large payloads).

Example: Why position:absolute sometimes moves relative to wrong ancestor?

- Because absolute is positioned relative to the **nearest ancestor that has a position other than static**. If none exist, it uses the viewport. Set parent position: relative to anchor.

---

15. DOM & Real-World Tasks (how to perform common quiz tasks)

- **Add an element:** createElement + appendChild.
- **Remove an element:** remove() or parentNode.removeChild(child).
- **Update text:** textContent or innerHTML (innerHTML risk: XSS).
- **Add/remove CSS class:** classList.add('x'), classList.remove('x'), classList.toggle('x').
- **Form validation:** onsubmit event, use e.preventDefault() to stop submission if invalid. Regular expressions for pattern matching.

---

16. Memory Aids & Final Tips

- **Remember the request lifecycle:** DNS → TCP → TLS → HTTP request → server logic → response → render.
- **Status code families:** 2xx good, 3xx redirect, 4xx client, 5xx server.
- **HTML semantics = accessibility + SEO.** Use <main>, <article>, <nav>.
- **Prefer let/const over var.** Use ===.
- **box-sizing: border-box** simplifies layout math.
- **Use external CSS for site consistency** and maintainability.
- **Always validate inputs server-side** — client-side validation is only UX convenience.

---

17. Quick Reference Code Snippets (paste into notes)

Sticky header

header { position: sticky; top: 0; z-index: 1000; background: #fff; }

JS: add event listener

document.getElementById('addBtn').addEventListener('click', addToCart);

Form: file upload

<form action="/upload" method="POST" enctype="multipart/form-data">

  <input type="file" name="avatar" required>

  <input type="submit" value="Upload">

</form>

Select only text inputs with CSS

input[type="text"] { border: 1px solid #ccc; }

18. Practice MCQs — explain one example fully (model)

Q: What does parseInt("20 mins") return?
Answer: 20.
Explanation: parseInt parses from the start of the string until encountering a non-numeric character, then returns the parsed integer. If the string starts with non-numeric characters, it returns NaN.