t of Computer Science & Engineering II Year / C Section / IT

Unit	Title	Periods
I	Website Basics	9
II	Web Designing	9
III	Client-Side Processing and Scripting	9
IV	TypeScript	9
V	Servlets and Database Connectivity	9

UNIT I

WEBSITE BASICS | 9 Peri

1.1 Internet Overview

The **Internet** is a global network of interconnected computers that communicate using standardised protocols. It originated from ARPANET (Advanced Research Projects Agency Network), developed in the late 1960s by the U.S. Department of Defense. Today, the Internet connects billions of devices worldwide, enabling services such as the World Wide Web, email, file sharing, video conferencing, and cloud computing.

> *Key Fact: The Internet is not the same as the World Wide Web. The Internet is the infrastructure (hardware + protocols), while the Web is one service running on that infrastructure.*

1.2 Fundamental Computer Network Concepts

A **computer network** is a collection of interconnected devices that share resources and information. Understanding networks is essential for web development.

Network Types

LAN	Local Area Network – covers a small area like a home, office, or campus. High speed, low latency.
WAN	Wide Area Network – spans large geographical areas; the Internet is the world's largest WAN.
MAN	Metropolitan Area Network – city-wide network, larger than LAN but smaller than WAN.
PAN	Personal Area Network – short-range (e.g., Bluetooth devices around a single person).

Network Topologies

- **Bus:** All nodes share a single communication line. Simple but a single break disrupts all.
- **Star:** All nodes connect to a central hub/switch. Easy to add nodes; hub failure affects all.
- **Ring:** Each node connects to exactly two others, forming a ring. Data travels in one direction.
- **Mesh:** Every node connects to every other node. Highly reliable but expensive.
- **Hybrid:** Combination of two or more topologies.

OSI Model

The **OSI (Open Systems Interconnection)** model defines seven layers of network communication:

Layer	Name	Function
7	Application	User interface, HTTP, FTP, SMTP
6	Presentation	Data formatting, encryption, compression
5	Session	Managing sessions between applications
4	Transport	Reliable delivery – TCP/UDP
3	Network	Routing and IP addressing
2	Data Link	MAC addressing, framing, error detection
1	Physical	Cables, signals, bits on the wire

1.3 Web Protocols

Protocols are agreed-upon rules that govern how data is transmitted across a network. Key web protocols include:

HTTP	HyperText Transfer Protocol – the foundation of data communication on the Web. Operates on port 80.
HTTPS	HTTP Secure – HTTP with SSL/TLS encryption. Operates on port 443. Essential for secure sites.
FTP	File Transfer Protocol – used to transfer files between client and server (ports 20/21).
SMTP	Simple Mail Transfer Protocol – handles outgoing email (port 25/587).
POP3	Post Office Protocol v3 – retrieves email from a server (port 110).
IMAP	Internet Message Access Protocol – manages email on the server (port 143).
DNS	Domain Name System – translates domain names to IP addresses (port 53).
TCP/IP	The core suite of protocols enabling Internet communication.

HTTP Request-Response Cycle

Every web interaction follows a request-response model:

- 1. User types a URL in the browser.
- 2. Browser sends an HTTP GET request to the server.
- 3. Server processes the request and returns an HTTP response (HTML, JSON, etc.).

- 4. Browser renders the response content.

Common HTTP Methods

Method	Purpose	Example Use
GET	Retrieve data	Fetch a webpage
POST	Submit data	Login form submission
PUT	Update/replace	Update user profile
DELETE	Delete a resource	Delete a record
PATCH	Partial update	Update one field

1.4 URL and Domain Name

A **URL (Uniform Resource Locator)** is the complete address used to access a resource on the Internet.

```
https://www.example.com:8080/path/page.html?id=5#section
| | | | | | |
scheme sub domain port path query fragment
```

Scheme	Protocol used: http, https, ftp, mailto, etc.
Sub-domain	Prefix like 'www'; can have multiple (e.g., mail.google.com).
Domain	Human-readable address (e.g., google.com).
Port	Optional; default is 80 (HTTP) or 443 (HTTPS).
Path	Specific resource location on the server.
Query	Key-value pairs for search/filtering (?key=value&key2;=val2).
Fragment	Anchor to a specific section of the page (#section).

Domain Name System (DNS)

DNS translates human-friendly domain names (like www.google.com) into machine-friendly IP addresses (like 142.250.190.46). The process is called **DNS Resolution**:

- Browser checks local cache for the IP address.
- If not found, query is sent to a Recursive Resolver (usually your ISP).
- Resolver queries Root Nameservers, then TLD Nameservers (e.g., .com).
- Finally, the Authoritative Nameserver returns the IP address.
- Browser connects to that IP address.

1.5 Web Browsers and Web Servers

Web Browsers

A **web browser** is a software application that retrieves, interprets, and displays web content. Popular browsers include Google Chrome, Mozilla Firefox, Microsoft Edge, Apple Safari, and Opera.

Core browser components:

User Interface	Address bar, buttons, tabs, bookmarks – everything except the viewport.
Browser Engine	Bridges the UI and rendering engine.
Rendering Engine	Parses HTML and CSS to display content (e.g., Blink in Chrome, Gecko in Firefox).
Networking	Handles HTTP/HTTPS requests and responses.
JS Engine	Executes JavaScript (e.g., V8 in Chrome).
Data Storage	Cookies, localStorage, IndexedDB, cache.

Web Servers

A **web server** is a computer that stores web content and serves it to clients upon request. Popular web servers include:

- **Apache HTTP Server** – open-source, highly configurable, widely used.
- **Nginx** – high-performance server, excellent for serving static files and as a reverse proxy.
- **Microsoft IIS** – Internet Information Services, integrated with Windows Server.
- **Node.js (http module)** – JavaScript runtime that can act as a web server.

1.6 Working Principle of a Website

When a user visits a website, the following sequence of events occurs:

Step 1	User enters a URL in the browser.
Step 2	DNS resolution converts the domain name to an IP address.
Step 3	Browser establishes a TCP connection to the server (3-way handshake).
Step 4	Browser sends an HTTP/HTTPS request to the server.
Step 5	Server processes the request (may query a database).
Step 6	Server sends back an HTTP response with HTML, CSS, JS files.
Step 7	Browser renders the page using its rendering engine.

Step 8	JavaScript executes and may make additional requests (AJAX).

1.7 Creating a Website

Building a basic website involves several stages:

- **Planning:** Define purpose, target audience, sitemap, and content.
- **Designing:** Create wireframes and mockups (tools: Figma, Adobe XD).
- **Development:** Write HTML (structure), CSS (style), and JavaScript (behaviour).
- **Testing:** Cross-browser testing, responsiveness, accessibility checks.
- **Deployment:** Upload files to a hosting server via FTP or CI/CD pipeline.
- **Maintenance:** Regular updates, security patches, performance optimisation.

```
<!DOCTYPE html>
<html lang='en'>
<head>
<meta charset='UTF-8'>
<title>My First Website</title>
<link rel='stylesheet' href='style.css'>
</head>
<body>
<h1>Hello, World!</h1>
<script src='app.js'></script>
</body>
</html>
```

1.8 Client-Side and Server-Side Scripting

Aspect	Client-Side Scripting	Server-Side Scripting
Execution	Runs in the user's browser	Runs on the web server
Languages	HTML, CSS, JavaScript	PHP, Python, Java, Node.js, Ruby
Speed	Faster (no server trip needed)	Depends on server load
Security	Source code visible to users	Logic hidden from users
DB Access	Cannot directly access database	Can query databases directly
Examples	Form validation, animations	Authentication, data processing

> *Summary: Unit I covers the foundational knowledge required to understand how the web works — from networks and protocols to browsers, servers, and scripting approaches.*

UNIT II

WEB DESIGNING | 9 Peri

2.1 HTML – Form Elements

HTML forms collect user input and submit it to a server. A form is created with the **<form>** element and contains various input controls.

```
<form action='/submit' method='POST'>
<label for='name'>Name:</label>
<input type='text' id='name' name='name' required>
<input type='submit' value='Send'>
</form>
```

Form Attributes

action	URL where form data is submitted.
method	HTTP method: GET or POST.
enctype	Encoding type – important for file uploads (multipart/form-data).
novalidate	Disables browser-built-in validation.
autocomplete	Enables/disables browser autocomplete.

2.2 Input Types and Media Elements

HTML5 Input Types

Type	Description	Example
text	Single-line text input	<input type="text">
password	Masked text input	<input type="password">
email	Validates email format	<input type="email">
number	Numeric input with min/max	<input type="number" min="0">
range	Slider control	<input type="range">
date	Date picker	<input type="date">
color	Color picker	<input type="color">
file	File upload control	<input type="file">
checkbox	Multi-select option	<input type="checkbox">
radio	Single-select from group	<input type="radio">

tel	Telephone number	<input type="tel">
url	Validates URL format	<input type="url">
search	Search field	<input type="search">
hidden	Hidden field (sent with form)	<input type="hidden">

Media Elements

HTML5 introduced native media elements that play audio and video without plugins:

```
<!-- Audio -->
<audio controls>
<source src='audio.mp3' type='audio/mpeg'>
<source src='audio.ogg' type='audio/ogg'>
Your browser does not support audio.
</audio>

<!-- Video -->
<video width='640' height='360' controls autoplay muted>
<source src='video.mp4' type='video/mp4'>
<track src='captions.vtt' kind='subtitles' srclang='en'>
</video>
```

2.3 CSS3 – Selectors

CSS selectors define which HTML elements a rule set applies to. CSS3 introduced powerful new selector types:

Universal (*)	Selects all elements.
Type (h1)	Selects all <h1> elements.
Class (.box)	Selects elements with class='box'.
ID (#header)	Selects the element with id='header'.
Attribute ([type])	Selects elements with a specific attribute.
[attr=val]	Exact attribute value match.
[attr^=val]	Attribute value starts with val.
[attr$=val]	Attribute value ends with val.
[attr*=val]	Attribute value contains val.
:hover	Pseudo-class – element when hovered.
:nth-child(n)	Selects nth child of parent.

::before / ::after	Pseudo-elements for inserting content.
> (child)	Direct child combinator.
~ (general sibling)	All siblings after an element.
+ (adjacent sibling)	Immediately following sibling.

2.4 CSS3 Box Model

Every HTML element is treated as a rectangular box. The CSS Box Model describes the space around content:

```
/* Box Model diagram */
+-----------------------------+ <- Margin (transparent, outside border)
| +-------------------------+ |
| | Border | |
| | +-------------------+ | |
| | | Padding | | |
| | | +-------------+ | | |
| | | | Content | | | |
| | | +-------------+ | | |
| | +-------------------+ | |
| +-------------------------+ |
+-----------------------------+

box-sizing: border-box; /* width includes padding + border */
box-sizing: content-box; /* width = content only (default) */
```

2.5 Backgrounds and Borders

```
/* Backgrounds */
background-color: #3498db;
background-image: url('bg.jpg');
background-size: cover; /* cover | contain | auto */
background-repeat: no-repeat;
background-position: center center;
background-attachment: fixed; /* parallax effect */

/* CSS Gradient */
background: linear-gradient(135deg, #1a3c6e, #2980b9);
background: radial-gradient(circle, #fff, #000);

/* Borders */
border: 2px solid #333;
border-radius: 8px; /* rounded corners */
border-image: url('frame.png') 30 round;
```

2.6 Text Effects and Animations

```
/* Text Effects */
text-shadow: 2px 2px 5px rgba(0,0,0,0.4);
text-transform: uppercase;
letter-spacing: 2px;
word-spacing: 5px;
text-overflow: ellipsis;
@font-face { font-family: 'MyFont'; src: url('font.woff2'); }

/* CSS Transitions */
transition: background-color 0.3s ease, transform 0.3s ease;
.btn:hover { background-color: #e67e22; transform: scale(1.05); }

/* CSS Keyframe Animations */
@keyframes slideIn {
from { transform: translateX(-100%); opacity: 0; }
to { transform: translateX(0); opacity: 1; }
}
.banner { animation: slideIn 0.6s ease forwards; }

/* Animation Properties */
animation-name: slideIn;
animation-duration: 0.6s;
animation-timing-function: ease-in-out;
animation-delay: 0.2s;
animation-iteration-count: infinite;
animation-direction: alternate;
```

2.7 Multiple Column Layout

CSS Multi-column layout allows content to flow automatically into multiple columns, similar to a newspaper:

```
.article {
column-count: 3; /* Number of columns */
column-gap: 2em; /* Space between columns */
column-rule: 1px solid #ccc; /* Line between columns */
column-width: 200px; /* Minimum column width */
}
.article h2 {
column-span: all; /* Span across all columns */
}
```

2.8 User Interface – Flexbox and Grid

Flexbox

```
.container {
display: flex;
flex-direction: row; /* row | column | row-reverse */
justify-content: center; /* flex-start | center | space-between */
align-items: center; /* flex-start | center | stretch */
```

```
flex-wrap: wrap; /* allow wrapping */
gap: 16px;
}
.item { flex: 1; } /* grow to fill available space */
```

CSS Grid

```
.grid {
display: grid;
grid-template-columns: repeat(3, 1fr); /* 3 equal columns */
grid-template-rows: auto;
gap: 20px;
}
.header { grid-column: 1 / -1; } /* span all columns */
.sidebar { grid-row: 2 / 4; } /* span 2 rows */
```

Summary: Unit II covers the visual layer of web development – crafting HTML forms, using CSS3 selectors, the box model, backgrounds, borders, text effects, animations, and modern layout systems like Flexbox and Grid.

UNIT III CLIENT-SIDE PROCESSING AND SCRI

3.1 JavaScript Introduction

JavaScript (JS) is a lightweight, interpreted, multi-paradigm programming language that runs in the browser (and on the server via Node.js). It makes web pages interactive — handling user events, validating forms, fetching data, and updating the DOM.

```
// Inline in HTML
<script>console.log('Hello, JavaScript!');</script>

// External file (recommended)
<script src='app.js' defer></script>
```

3.2 Variables and Data Types

```
// Variable declarations
var x = 10; // function-scoped, hoisted (avoid in modern JS)
let y = 20; // block-scoped, reassignable
const PI = 3.14; // block-scoped, constant (cannot reassign)

// Primitive Data Types
let name = 'Alice'; // String
let age = 25; // Number
let price = 9.99; // Number (floats same type)
let isOpen = true; // Boolean
let val = null; // Null (intentional absence)
let undef; // Undefined
let sym = Symbol('id'); // Symbol (unique identifier)
let bigInt = 9007199254740991n; // BigInt

// Reference Types
let arr = [1, 2, 3]; // Array
let obj = { name: 'Alice', age: 25 }; // Object
let fn = function() { return 42; }; // Function

typeof 'hello' // 'string'
typeof 42 // 'number'
typeof true // 'boolean'
typeof undefined // 'undefined'
typeof null // 'object' (historical quirk)
```

3.3 Statements and Operators

Control Statements

```
// if-else
if (score >= 90) { grade = 'A'; }
else if (score >= 75) { grade = 'B'; }
else { grade = 'C'; }

// switch
switch(day) {
case 1: console.log('Mon'); break;
default: console.log('Other');
}

// Loops
for (let i = 0; i < 5; i++) { console.log(i); }
while (x > 0) { x--; }
do { x++; } while (x < 10);
for (let key in obj) { /* iterate object keys */ }
for (let val of arr) { /* iterate array values */ }
```

Operators

Arithmetic	+ - * / % ** (exponentiation) ++ --
Comparison	== != === !== > < >= <= (=== is strict equality)
Logical	&& (AND) || (OR) ! (NOT) ?? (nullish coalescing)
Assignment	= += -= *= /= %= **= &&= ||= ??=
Bitwise	& | ^ ~ << >> >>>
Ternary	condition ? valueIfTrue : valueIfFalse
Spread/Rest	... (spread elements or collect rest parameters)
Optional ?.	obj?.prop – safe access, returns undefined if null/undefined

3.4 Functions

```
// Function Declaration (hoisted)
function add(a, b) { return a + b; }

// Function Expression
const multiply = function(a, b) { return a * b; };

// Arrow Function (ES6+)
const square = (n) => n * n;
const greet = name => `Hello, ${name}!`; // template literal

// Default Parameters
function power(base, exp = 2) { return base ** exp; }

// Rest Parameters
```

```
function sum(...nums) { return nums.reduce((a,b) => a+b, 0); }

// Immediately Invoked Function Expression (IIFE)
(function() { console.log('Runs immediately!'); })();

// Higher-Order Functions
const doubled = [1,2,3].map(n => n * 2); // [2,4,6]
const evens = [1,2,3,4].filter(n => n%2===0); // [2,4]
const total = [1,2,3].reduce((acc,n) => acc+n, 0); // 6
```

3.5 Objects and Arrays

```
// Object literal
const student = {
name: 'Alice',
marks: [85, 90, 78],
greet() { return `Hi, I'm ${this.name}`; }
};
console.log(student.name); // dot notation
console.log(student['marks'][0]); // bracket notation

// Destructuring
const { name, marks } = student;
const [first, , third] = marks;

// Spread
const arr1 = [1,2], arr2 = [...arr1, 3, 4]; // [1,2,3,4]
const obj2 = { ...student, age: 20 };

// Array Methods
arr.push(5); arr.pop(); arr.shift(); arr.unshift(0);
arr.slice(1, 3); // non-mutating subset
arr.splice(1, 2, 'x'); // mutating: remove 2, insert 'x'
arr.indexOf(3); // find index
arr.includes(3); // true/false
arr.sort((a,b) => a-b); // ascending sort
arr.find(n => n > 3); // first match
arr.every(n => n > 0); // all match?
arr.some(n => n > 3); // any match?
```

3.6 Built-in Objects

Math	Math.round(), Math.floor(), Math.ceil(), Math.random(), Math.max(), Math.min(), Math.PI
Date	new Date(), .getFullYear(), .getMonth(), .getDate(), .getTime(), Date.now()

String	.length, .toUpperCase(), .trim(), .split(), .replace(), .includes(), .startsWith(), .padStart()
Number	Number.isInteger(), Number.parseFloat(), .toFixed(), Number.isNaN()
JSON	JSON.stringify(obj), JSON.parse(str) – serialise/deserialise objects
RegExp	/pattern/flags, .test(), .exec(), str.match(), str.replace()
Promise	new Promise((resolve, reject) => {}), .then(), .catch(), .finally()
Map/Set	Map – key-value with any key type; Set – unique values only

3.7 Regular Expressions

```
// Syntax: /pattern/flags
// Flags: g (global), i (case-insensitive), m (multiline)

const emailRegex = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
emailRegex.test('user@example.com'); // true

// Common patterns
/\d+/ // one or more digits
/\w+/ // word characters (a-z, A-Z, 0-9, _)
/\s/ // whitespace
/^Hello/ // starts with Hello
/world$/ // ends with world
/[aeiou]/gi // match all vowels (case-insensitive)

'hello world'.replace(/world/, 'JS'); // 'hello JS'
'a1b2c3'.match(/\d/g); // ['1','2','3']
'one two three'.split(/\s+/); // ['one','two','three']
```

3.8 Exceptions and Event Handling

```
// Error Handling
try {
const data = JSON.parse(invalidJSON);
} catch (error) {
console.error('Parse error:', error.message);
} finally {
console.log('Always runs');
}

// Custom Error
throw new Error('Something went wrong');
throw new TypeError('Expected a string');

// DOM Event Handling
```

```
const btn = document.getElementById('myBtn');
btn.addEventListener('click', function(event) {
event.preventDefault(); // stop default action
event.stopPropagation(); // stop bubbling
console.log('Button clicked!');
});

// Common Events
// click, dblclick, mouseover, mouseout, keydown, keyup
// submit, change, input, focus, blur, load, DOMContentLoaded
```

3.9 Form Validation

```
function validateForm(e) {
e.preventDefault();
const name = document.getElementById('name').value.trim();
const email = document.getElementById('email').value.trim();
const emailRe = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
let errors = [];
if (!name) errors.push('Name is required.');
if (!emailRe.test(email)) errors.push('Invalid email.');
if (errors.length > 0) {
alert(errors.join('\n'));
} else {
document.querySelector('form').submit();
}
}
```

3.10 JavaScript Debuggers

Debugging is the process of finding and fixing errors (bugs) in code. JavaScript provides several tools and techniques:

console.log()	Print values to the browser console for inspection.
console.error()	Output error messages in red in the console.
console.table()	Display arrays/objects as formatted tables.
debugger;	Pauses execution at that line when DevTools is open.
Breakpoints	Set in browser DevTools Sources panel – pause execution at a line.
Watch Expressions	Monitor variable values in real time in DevTools.
Call Stack	Shows the chain of function calls that led to the current point.
Network Tab	Inspect HTTP requests, responses, headers, and timing.

Linters (ESLint)	Static analysis tool that catches errors before runtime.

Summary: Unit III covers JavaScript from basics to advanced concepts – variables, types, operators, functions, objects, built-in APIs, regular expressions, exceptions, events, validation, and debugging.

UNIT IV

TYPESCRIPT | 9 Period

4.1 Introduction to TypeScript

TypeScript (TS) is an open-source, strongly-typed superset of JavaScript developed by Microsoft. It compiles down to plain JavaScript and adds optional static typing, interfaces, generics, and other features that improve large-scale application development.

> *TypeScript = JavaScript + Static Types + Modern Features. Any valid JS is valid TS.*

```
// Install TypeScript
npm install -g typescript

// Compile
tsc app.ts // produces app.js
tsc --watch app.ts // watch mode

// tsconfig.json (project configuration)
{ "compilerOptions": {
"target": "ES6",
"strict": true,
"outDir": "./dist"
}
}
```

4.2 TypeScript Basics – Data Types and Variables

```
// Explicit type annotations
let name: string = 'Alice';
let age: number = 25;
let isAdmin: boolean = false;
let score: number | string = 90; // Union type
let data: any = 'anything'; // Opt out of type checking
let undef: undefined = undefined;
let nothing: null = null;

// Arrays
let nums: number[] = [1, 2, 3];
let strs: Array = ['a', 'b'];

// Tuple (fixed-length, typed array)
let coord: [number, number] = [10.5, 20.3];

// Enum
enum Direction { Up, Down, Left, Right }
let move: Direction = Direction.Up; // move = 0
```

```
// Literal Types
let status: 'active' | 'inactive' | 'pending' = 'active';

// Type Alias
type ID = string | number;
let userId: ID = 'usr_42';

// void and never
function log(msg: string): void { console.log(msg); }
function fail(msg: string): never { throw new Error(msg); }
```

4.3 Destructuring and Spread

```
// Array Destructuring
const [x, y, ...rest]: number[] = [1, 2, 3, 4, 5];
// x=1, y=2, rest=[3,4,5]

// Object Destructuring with types
const { name, age }: { name: string; age: number } = person;

// Default values in destructuring
const { host = 'localhost', port = 3000 } = config;

// Spread operator
const arr1: number[] = [1, 2];
const arr2: number[] = [...arr1, 3, 4]; // [1,2,3,4]

const obj1 = { a: 1 };
const obj2 = { ...obj1, b: 2 }; // { a:1, b:2 }

// Function rest parameters
function total(...nums: number[]): number {
return nums.reduce((a, b) => a + b, 0);
}
```

4.4 Working with Classes

```
class Animal {
// Access modifiers: public (default), private, protected, readonly
private name: string;
protected sound: string;
readonly id: number;

constructor(name: string, sound: string) {
this.name = name;
this.sound = sound;
this.id = Math.random();
}
```

```
speak(): string { return `${this.name} says ${this.sound}`; }

// Getter / Setter
get animalName(): string { return this.name; }
set animalName(v: string) { this.name = v; }
}

// Inheritance
class Dog extends Animal {
breed: string;
constructor(name: string, breed: string) {
super(name, 'Woof');
this.breed = breed;
}
// Method overriding
speak(): string { return super.speak() + '!'; }
}

const dog = new Dog('Rex', 'Labrador');
console.log(dog.speak()); // Rex says Woof!
```

4.5 Interfaces

```
// Interface definition
interface User {
id: number;
name: string;
email?: string; // optional property
readonly createdAt: Date; // read-only
}

// Implementing an interface
class Admin implements User {
id: number;
name: string;
createdAt: Date;
role: string = 'admin';
constructor(id: number, name: string) {
this.id = id; this.name = name;
this.createdAt = new Date();
}
}

// Interface extending another
interface SuperAdmin extends User {
permissions: string[];
}

// Function interface
interface MathFn {
```

```
(x: number, y: number): number;
}
const add: MathFn = (a, b) => a + b;
```

4.6 Generics

```
// Generic function
function identity(arg: T): T { return arg; }
identity('hello'); // returns 'hello'
identity(42); // returns 42

// Generic interface
interface ApiResponse {
data: T;
status: number;
message: string;
}
const res: ApiResponse = await fetch('/users');

// Generic class
class Stack {
private items: T[] = [];
push(item: T): void { this.items.push(item); }
pop(): T | undefined { return this.items.pop(); }
get size(): number { return this.items.length; }
}

// Constraints
function getLength(arg: T): number {
return arg.length;
}
```

4.7 Modules and Namespaces

```
// math.ts – exporting
export function add(a: number, b: number): number { return a + b; }
export const PI = 3.14159;
export default class Calculator { /* ... */ }

// app.ts – importing
import Calculator, { add, PI } from './math';
import * as MathUtils from './math';

// Namespace (for organising code within a file)
namespace Validation {
export interface Validator { isValid(s: string): boolean; }
export class EmailValidator implements Validator {
isValid(s: string): boolean {
return /^[^@]+@[^@]+$/.test(s);
```

```
}
}
}
const v = new Validation.EmailValidator();
```

4.8 Functions, Loops, and Collections

```
// Typed function signatures
const greet = (name: string, greeting: string = 'Hello'): string =>
`${greeting}, ${name}!`;

// Function overloads
function format(x: string): string;
function format(x: number): string;
function format(x: any): string { return String(x); }

// for...of with type
const nums: number[] = [1, 2, 3];
for (const n of nums) { console.log(n * 2); }

// Map collection
const map = new Map();
map.set('a', 1); map.get('a'); // 1
map.has('a'); map.delete('a');
for (const [key, val] of map) { console.log(key, val); }

// Set collection
const set = new Set([1, 2, 2, 3]); // {1, 2, 3}
set.add(4); set.has(2); set.size; // 4

// WeakMap / WeakRef (for memory-sensitive scenarios)
const wm = new WeakMap();
```

> *Summary: TypeScript enhances JavaScript with static typing, interfaces, generics, access modifiers, modules, and namespaces – enabling safer, more maintainable large-scale applications.*

UNIT V

SERVLETS AND DATABASE CONNECT

5.1 Servlets – Introduction

A **Java Servlet** is a server-side Java program that handles HTTP requests and generates dynamic web content. Servlets run inside a **Servlet Container** (also called a Web Container), such as Apache Tomcat, GlassFish, or JBoss.

> *Servlets replaced older CGI (Common Gateway Interface) scripts, offering better performance, portability, and integration with the Java EE ecosystem.*

Advantages of Servlets

- **Platform Independent:** Java runs on any OS with a JVM.
- **Persistent:** Servlets stay in memory between requests (unlike CGI).
- **Efficient:** A single servlet instance handles multiple requests via threads.
- **Secure:** Java's security model protects against common vulnerabilities.
- **Rich API:** Access to all Java EE libraries (JDBC, JPA, EJB, etc.).

5.2 Java Servlet Architecture

The Servlet API is defined in the **javax.servlet** and **javax.servlet.http** packages. The core hierarchy is:

Servlet (interface)	Root interface – defines init(), service(), destroy().
GenericServlet (abstract)	Protocol-independent implementation of Servlet.
HttpServlet (abstract)	Extends GenericServlet; provides doGet(), doPost(), doPut(), doDelete(), etc.
HttpServletRequest	Encapsulates client request data – parameters, headers, session.
HttpServletResponse	Encapsulates server response – status codes, headers, output stream.
ServletConfig	Per-servlet configuration (init parameters from web.xml).
ServletContext	Application-wide context shared by all servlets.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

@WebServlet('/hello') // Annotation-based mapping
public class HelloServlet extends HttpServlet {
```

```
@Override
protected void doGet(HttpServletRequest req,
HttpServletResponse res)
throws ServletException, IOException {

res.setContentType('text/html;charset=UTF-8');
PrintWriter out = res.getWriter();
String name = req.getParameter('name');
out.println('');
out.println('Hello, ' + name + '!');
out.println('');
}
}
```

5.3 Servlet Life Cycle

The servlet life cycle is managed by the Servlet Container and consists of four phases:

Phase	Method	Description
1. Loading	—	Container loads the servlet class.
2. Instantiation	new ()	Container creates one instance of the servlet.
3. Initialisation	init()	Called once – setup resources, read init params.
4. Request Handling	service()	Called for every request; dispatches to doGet(), doPost(), etc.
5. Destruction	destroy()	Called once before servlet is removed – release resources.

5.4 Form GET and POST Actions

GET vs POST

Feature	GET	POST
Data Location	Appended to URL as query string	Sent in HTTP request body
Visibility	Visible in browser address bar	Hidden from URL
Security	Less secure (data in URL logs)	More secure for sensitive data
Data Size	Limited (~2000 chars)	No practical limit
Cacheable	Yes	No
Use Case	Search, filter, read operations	Login, form submission, file upload

Idempotent	Yes (same result on repeat)	No

5.5 Sessions and Cookies

HTTP is a stateless protocol. Sessions and cookies allow maintaining state across multiple requests.

Cookies

```
// Set a cookie
Cookie cookie = new Cookie('username', 'alice');
cookie.setMaxAge(60 * 60 * 24); // 1 day in seconds
cookie.setPath('/');
cookie.setSecure(true); // HTTPS only
cookie.setHttpOnly(true); // no JS access
response.addCookie(cookie);

// Read cookies
Cookie[] cookies = request.getCookies();
for (Cookie c : cookies) {
if ('username'.equals(c.getName())) {
String user = c.getValue();
}
}

// Delete a cookie
cookie.setMaxAge(0);
response.addCookie(cookie);
```

Sessions

```
// Create / retrieve session
HttpSession session = request.getSession(); // creates if not exists
HttpSession session = request.getSession(false); // returns null if not exists

// Store and retrieve attributes
session.setAttribute('user', userObject);
User u = (User) session.getAttribute('user');

// Session properties
session.getId(); // unique session ID
session.setMaxInactiveInterval(1800); // 30 minutes timeout

// Invalidate (logout)
session.invalidate();
```

5.6 Database Connectivity – JDBC

JDBC (Java Database Connectivity) is the standard Java API for connecting to relational databases. It provides a uniform interface regardless of the database vendor.

JDBC Architecture

JDBC API	java.sql and javax.sql packages – the interfaces your code calls.
JDBC Driver Manager	Manages a list of database drivers.
JDBC Driver	Vendor-specific implementation (MySQL Connector/J, Oracle JDBC, etc.).
Database	The actual RDBMS (MySQL, PostgreSQL, Oracle, SQLite, etc.).

JDBC Driver Types

- **Type 1 – JDBC-ODBC Bridge:** Maps JDBC calls to ODBC (deprecated in Java 8).
- **Type 2 – Native API:** Uses vendor-specific native libraries.
- **Type 3 – Network Protocol:** Sends to middleware; middleware talks to DB.
- **Type 4 – Pure Java (Thin Driver):** Direct DB protocol implementation – most common.

JDBC Steps

```
import java.sql.*;

// Step 1: Load Driver (not needed in JDBC 4+)
Class.forName('com.mysql.cj.jdbc.Driver');

// Step 2: Establish Connection
String url = 'jdbc:mysql://localhost:3306/mydb';
Connection conn = DriverManager.getConnection(url, 'root', 'password');

// Step 3: Create Statement
Statement stmt = conn.createStatement();

// Step 4: Execute Query
ResultSet rs = stmt.executeQuery('SELECT * FROM students');

// Step 5: Process Results
while (rs.next()) {
int id = rs.getInt('id');
String name = rs.getString('name');
double gpa = rs.getDouble('gpa');
System.out.println(id + ' | ' + name + ' | ' + gpa);
}

// Step 6: Close Resources
rs.close(); stmt.close(); conn.close();
```

PreparedStatement (Prevents SQL Injection)

```
String sql = 'INSERT INTO students (name, gpa) VALUES (?, ?)';
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1, 'Bob');
ps.setDouble(2, 3.75);
```

```
int rowsInserted = ps.executeUpdate(); // returns affected rows

// Parameterised SELECT
String query = 'SELECT * FROM students WHERE id = ?';
PreparedStatement ps2 = conn.prepareStatement(query);
ps2.setInt(1, 5);
ResultSet rs = ps2.executeQuery();
```

5.7 Simple Interactive Database Application

A complete student registration web application using Servlets and JDBC:

```
/* StudentServlet.java */
@WebServlet('/students')
public class StudentServlet extends HttpServlet {

private static final String DB_URL =
'jdbc:mysql://localhost:3306/school';

@Override
protected void doPost(HttpServletRequest req,
HttpServletResponse res)
throws ServletException, IOException {

String name = req.getParameter('name');
String gpa = req.getParameter('gpa');

try (Connection conn =
DriverManager.getConnection(DB_URL,'root','pw');
PreparedStatement ps = conn.prepareStatement(
'INSERT INTO students(name,gpa) VALUES(?,?)')) {

ps.setString(1, name);
ps.setDouble(2, Double.parseDouble(gpa));
ps.executeUpdate();
res.sendRedirect('/students?msg=Added');

} catch (SQLException e) {
throw new ServletException(e);
}
}

@Override
protected void doGet(HttpServletRequest req,
HttpServletResponse res)
throws ServletException, IOException {

List students = new ArrayList<>();
try (Connection conn =
DriverManager.getConnection(DB_URL,'root','pw');
Statement st = conn.createStatement();
```

```
ResultSet rs = st.executeQuery(
'SELECT name, gpa FROM students ORDER BY gpa DESC')) {

while (rs.next())
students.add(rs.getString('name') + ' - ' + rs.getDouble('gpa'));

} catch (SQLException e) {
throw new ServletException(e);
}
req.setAttribute('students', students);
req.getRequestDispatcher('/students.jsp')
.forward(req, res);
}
}
```

JDBC Statement Types Comparison

Type	Class	Best For	SQL Injection Safe?
Statement	Statement	Static queries	No
Prepared	PreparedStatement	Repeated queries with params	Yes
Callable	CallableStatement	Stored procedures	Yes

Summary: Unit V covers server-side Java development with Servlets – life cycle, GET/POST handling, sessions, cookies, and full database connectivity using JDBC for building dynamic, data-driven web applications.

APPENDIX

Quick Reference & Key Te

HTTP Status Codes

Code	Meaning
200 OK	Request succeeded.
201 Created	Resource created successfully.
301 Moved Permanently	URL has permanently changed.
302 Found	Temporary redirect.
400 Bad Request	Malformed request syntax.
401 Unauthorised	Authentication required.
403 Forbidden	Server refuses to authorise.
404 Not Found	Resource does not exist on server.
500 Internal Server Error	Server encountered unexpected error.
503 Service Unavailable	Server is temporarily down.

CSS Units Reference

px	Absolute pixel – fixed size regardless of screen.
em	Relative to parent element's font size.
rem	Relative to root element's font size – predictable.
%	Percentage of the containing element.
vw	1% of viewport width.
vh	1% of viewport height.
fr	Fraction of available grid space.
ch	Width of the '0' character in current font.

JavaScript ES6+ Feature Summary

let / const	Block-scoped variable declarations.
Arrow Functions	Concise function syntax; lexical 'this'.
Template Literals	Backtick strings with embedded expressions.
Destructuring	Extract values from arrays/objects easily.
Spread / Rest (...)	Expand arrays/objects or collect arguments.
Default Parameters	Function parameters with default values.
Classes	Syntactic sugar over prototype-based inheritance.
Modules (import/export)	Native module system.
Promises / async-await	Asynchronous code without callback hell.
Optional Chaining (?.)	Safe property access.
Nullish Coalescing (??)	Fallback when value is null/undefined.
Map / Set	New collection types with clean APIs.

TypeScript vs JavaScript – Quick Comparison

Feature	JavaScript	TypeScript
Typing	Dynamic	Static (optional)
Compilation	None (interpreted)	Compiles to JS
Interfaces	No	Yes
Generics	No	Yes
Enums	Workarounds	Built-in
IDE Support	Good	Excellent (IntelliSense)
Error Catching	Runtime	Compile time
File Extension	.js	.ts / .tsx

Important Definitions – Glossary

API	Application Programming Interface – a set of rules for software components to communicate.

DOM	Document Object Model – the browser's tree representation of the HTML document.
AJAX	Asynchronous JavaScript And XML – technique to fetch data without full page reload.
REST	REpresentational State Transfer – an architectural style for designing web APIs.
JSON	JavaScript Object Notation – lightweight data interchange format.
CORS	Cross-Origin Resource Sharing – browser security policy for cross-domain requests.
SSL/TLS	Protocols for encrypted communication (the 'S' in HTTPS).
CDN	Content Delivery Network – distributed servers for faster content delivery.
MVC	Model-View-Controller – a design pattern for organising web application code.
ORM	Object-Relational Mapping – maps database rows to Java/Python objects.
WAR	Web Application Archive – a packaged Java web application (.war file).
Servlet Container	A server that manages the lifecycle of servlets (e.g., Apache Tomcat).

CTICE QUESTIONS

Unit-wise Question Bank

Unit I – Website Basics

Part A – 2 Mark Questions

1. Define the Internet and distinguish it from the World Wide Web.
2. What is a URL? Label its components.
3. List four types of computer networks.
4. What is DNS? Explain DNS Resolution in two sentences.
5. Differentiate HTTP and HTTPS.
6. What is client-side scripting? Give two examples.
7. Define a web server and name two popular web servers.

8. What does the HTTP status code 404 indicate?

Part B – 5 Mark Questions

1. Explain the seven layers of the OSI model with their functions.
2. Describe the request-response cycle of HTTP with a diagram.
3. Compare client-side and server-side scripting across five parameters.
4. Explain the components of a web browser in detail.
5. What are web protocols? Explain HTTP, FTP, and SMTP.

Part C – 10 Mark Questions

1. Explain the working principle of a website from URL entry to page render. Include DNS resolution, TCP handshake, and HTTP exchange.
2. Describe the fundamental computer network concepts: types of networks, topologies, and the OSI model.

Unit II – Web Designing

Part A – 2 Mark Questions

1. What is the HTML <form> element? Name two of its attributes.
2. List any five HTML5 input types introduced in HTML5.
3. What is the CSS Box Model? Name its four components.
4. Differentiate class selector and ID selector in CSS.
5. What is the purpose of box-sizing: border-box?
6. Write CSS to create a linear gradient background.
7. What is the difference between transition and animation in CSS?
8. What is CSS Flexbox? Name two justify-content values.

Part B – 5 Mark Questions

1. Explain CSS3 selectors with examples for five different selector types.
2. Describe CSS animations using @keyframes with a practical example.
3. Explain CSS Grid layout with a two-column, three-row example.
4. What are HTML5 media elements? Write code for embedding video with controls.
5. Describe CSS background properties with examples of color, gradient, and image.

Part C – 10 Mark Questions

1. Design a responsive web form using HTML5 input types and CSS3 styling. Include at least six different input types, custom styling, and hover effects.

2. Explain the CSS Box Model, Flexbox, and Grid Layout in detail with code examples showing their differences and use cases.

Unit III – Client-Side Processing and Scripting

Part A – 2 Mark Questions

1. What is JavaScript? How is it included in an HTML page?

2. Differentiate var, let, and const in JavaScript.

3. What are JavaScript primitive data types? List all seven.

4. Write a JavaScript arrow function to find the square of a number.

5. What is the difference between == and === in JavaScript?

6. What is an event listener? Give one example.

7. What does JSON.stringify() do?

8. Name four JavaScript debugging tools/techniques.

Part B – 5 Mark Questions

1. Explain JavaScript arrays with five useful array methods and examples.

2. Write a JavaScript function for client-side form validation (name, email, phone).

3. Explain Regular Expressions in JavaScript with five pattern examples.

4. What are JavaScript built-in objects? Explain Math and Date with examples.

5. Explain exception handling in JavaScript using try, catch, finally, and throw.

Part C – 10 Mark Questions

1. Explain JavaScript functions in detail – function declarations, expressions, arrow functions, higher-order functions, closures, and immediately invoked function expressions with code examples.

2. Describe the JavaScript event handling model. Explain event bubbling, capturing, preventDefault(), and stopPropagation() with practical examples.

Unit IV – TypeScript

Part A – 2 Mark Questions

1. What is TypeScript? How does it differ from JavaScript?

2. What is a TypeScript interface? Write a simple interface.

3. Differentiate 'any' and 'unknown' types in TypeScript.

4. What is a TypeScript enum? Write an example with three values.

5. What is the purpose of the 'readonly' modifier?

6. What are TypeScript generics? Give a one-line definition.

7. How do you compile a TypeScript file?

8. What is a Union type in TypeScript? Give an example.

Part B – 5 Mark Questions

1. Explain TypeScript classes with access modifiers, getters, setters, and inheritance.

2. Write a generic Stack class in TypeScript that supports push, pop, and peek.

3. Explain TypeScript modules – how to export and import functions, classes, and interfaces.

4. Describe destructuring and the spread operator in TypeScript with typed examples.

5. Explain TypeScript function overloads and default/rest parameters.

Part C – 10 Mark Questions

1. Explain TypeScript generics in detail: generic functions, generic interfaces, generic classes, and constraints. Provide code examples for each.

2. Design a TypeScript application modelling a library system with interfaces, classes (inheritance), generics for a typed collection, and proper module organisation.

Unit V – Servlets and Database Connectivity

Part A – 2 Mark Questions

1. What is a Java Servlet? Name two servlet containers.

2. List the five phases of the Servlet life cycle.

3. What is the difference between doGet() and doPost()?

4. What is a Cookie? How is its expiry set in Java?

5. What is an HttpSession? How do you invalidate it?

6. What does JDBC stand for? Name two JDBC driver types.

7. What is a PreparedStatement? Why is it preferred over Statement?

8. What is the purpose of the @WebServlet annotation?

Part B – 5 Mark Questions

1. Explain the Java Servlet architecture and the javax.servlet hierarchy.

2. Write Java code to create a cookie, retrieve it, and delete it.

3. Describe JDBC step-by-step with a complete code example for SELECT.

4. Compare GET and POST methods across six parameters.

5. Explain HttpSession management: creation, attribute storage, timeout, and invalidation.

Part C – 10 Mark Questions

1. Write a complete Servlet application that accepts student data via POST, stores it in a MySQL database using JDBC PreparedStatement, and displays all records via GET.

2. Explain the Servlet life cycle in detail. Describe the role of init(), service(), and destroy() with code, and explain how the container manages multiple concurrent requests.

XTENDED NOTES

Tips, Best Practices & Common

Web Development Best Practices

HTML Best Practices

- **Always declare DOCTYPE:** <!DOCTYPE html> ensures standards mode rendering.
- **Use semantic elements:** Prefer <header>, <nav>, <main>, <section>, <footer> over generic for accessibility and SEO.
- **Add alt text:** Every <img> must have a descriptive alt attribute.
- **Validate your HTML:** Use the W3C Markup Validator (validator.w3.org).
- **Minimise inline styles:** Keep styles in external CSS files for maintainability.
- **Use <label> with form controls:** Associate labels with inputs for accessibility.

CSS Best Practices

- **Mobile-first design:** Write styles for small screens first, then use media queries to scale up.
- **Use CSS custom properties (variables):** --primary-color: #1a3c6e for easy theming.
- **Avoid !important:** It makes debugging difficult; fix specificity instead.
- **Use shorthand properties:** margin: 10px 20px is cleaner than four separate declarations.
- **Organise stylesheets:** Group by component/section; use comments.
- **Prefer flexbox/grid over floats:** Modern layout tools are far more predictable.

JavaScript Best Practices

- **Always use strict mode:** 'use strict'; at the top catches silent errors.
- **Use const by default, let when needed:** Never use var in modern code.
- **Handle asynchronous code with async/await:** Cleaner than .then() chains.
- **Avoid global variables:** Use modules or IIFEs to encapsulate code.
- **Validate all inputs:** Both client-side (UX) and server-side (security).
- **Use === for comparisons:** Avoids type coercion surprises.
- **Write small, single-purpose functions:** Easier to test and reuse.

Common Mistakes to Avoid

Mistake	Area	Why It Matters / Fix
Using == instead of ===	JavaScript	== performs type coercion: 0 == '0' is true. Use === for predictable comparisons.

Not closing DB connections	JDBC	Always close ResultSet, Statement, and Connection in a finally block or use try-with-resources.
SQL Injection	JDBC	Never concatenate user input into SQL strings. Always use PreparedStatement.
Forgetting DOCTYPE	HTML	Without DOCTYPE, browsers enter quirks mode causing inconsistent rendering.
Using var in modern JS	JavaScript	var is function-scoped and hoisted; use let and const for predictable behaviour.
Missing alt on images	HTML/CSS	Breaks accessibility for screen readers and fails automated accessibility checks.
Hardcoding DB credentials	Java/JDBC	Store credentials in environment variables or a configuration file, not in source code.
Not sanitising output	Servlet	Always escape user data before writing to HTML to prevent XSS attacks.

Responsive Web Design Essentials

Responsive Web Design (RWD) ensures a website looks and works well on all screen sizes — from mobile phones to large desktop monitors.

```
/* Viewport meta tag – required for mobile responsiveness */
<meta name='viewport' content='width=device-width, initial-scale=1.0'>

/* Media Queries */
@media (max-width: 768px) {
.container { flex-direction: column; }
.sidebar { display: none; }
font-size: 16px;
}

@media (min-width: 769px) and (max-width: 1200px) {
.container { grid-template-columns: 1fr 2fr; }
}

/* Fluid images */
img { max-width: 100%; height: auto; }

/* Responsive typography */
h1 { font-size: clamp(1.5rem, 4vw, 3rem); }
```

Async JavaScript – Promises and async/await

```
// Callback (old-style – leads to 'callback hell')
fetch('/api/users', function(err, data) {
if (err) handleError(err);
else processData(data);
});

// Promise chain
fetch('/api/users')
.then(response => response.json())
.then(data => { console.log(data); })
.catch(error => { console.error(error); })
.finally(() => { hideSpinner(); });

// async/await (cleanest approach)
async function loadUsers() {
try {
const response = await fetch('/api/users');
if (!response.ok) throw new Error('HTTP ' + response.status);
const data = await response.json();
renderUsers(data);
} catch (error) {
console.error('Failed to load users:', error);
} finally {
hideSpinner();
}
}

// Parallel requests
const [users, posts] = await Promise.all([
fetch('/api/users').then(r => r.json()),
fetch('/api/posts').then(r => r.json()),
]);
```

Connection Pooling with JDBC

Creating a new database connection for every request is expensive. **Connection pooling** maintains a pool of reusable connections, dramatically improving performance.

```

<dependency>
<groupId>com.zaxxer</groupId>
<artifactId>HikariCP</artifactId>
<version>5.0.1</version>
</dependency>

// Configure HikariCP pool
HikariConfig config = new HikariConfig();
config.setJdbcUrl('jdbc:mysql://localhost:3306/mydb');
config.setUsername('root');
config.setPassword('password');
config.setMaximumPoolSize(10); // max 10 connections
config.setMinimumIdle(2); // keep 2 idle connections
```

```
config.setConnectionTimeout(30000); // 30 second timeout

HikariDataSource ds = new HikariDataSource(config);

// Borrow from pool
try (Connection conn = ds.getConnection();
PreparedStatement ps = conn.prepareStatement('SELECT * FROM users')) {
ResultSet rs = ps.executeQuery();
// process results...
} // connection automatically returned to pool
```

CSS Custom Properties (Variables)

```
:root {
--primary: #1a3c6e;
--secondary: #2980b9;
--accent: #e67e22;
--font-base: 16px;
--radius: 8px;
--shadow: 0 4px 12px rgba(0,0,0,0.15);
}

.card {
background: white;
border-radius: var(--radius);
box-shadow: var(--shadow);
padding: 1.5rem;
}

.btn-primary {
background-color: var(--primary);
color: white;
border-radius: var(--radius);
}

/* Override for dark theme */
@media (prefers-color-scheme: dark) {
:root { --primary: #2980b9; }
}
```

End of Learning Material. This document covers all five units of U23ITT43 – Web Technology, including theory, code examples, tables, practice questions, and extended notes. Good luck in your examinations!

DEEP DIVE

HTML5, CSS Techniques & Adva

HTML5 Semantic Elements

HTML5 introduced a rich set of semantic elements that give meaning to the page structure, improving accessibility, SEO, and code readability.

<header>	Introductory content for a page or section – logo, navigation.
<nav>	Block of navigation links.
<main>	The dominant content of the <body>; unique per page.
<article>	Self-contained, independently distributable content (blog post, news item).
<section>	Thematic grouping of content with a heading.
<aside>	Content tangentially related to main content – sidebar, callout.
<footer>	Footer for its nearest ancestor – copyright, links.
<figure>	Self-contained content referenced from main flow (image + caption).
<figcaption>	Caption for a <figure> element.
<time>	Machine-readable date/time: <time datetime='2025-01-15'>.
<mark>	Highlighted/relevant text.
<details>	Disclosure widget – content shown/hidden by user.
<summary>	Visible heading for a <details> element.
<dialog>	Modal or non-modal dialog box.
<progress>	Progress bar for a task.
<meter>	Scalar value within a known range.

CSS Pseudo-classes and Pseudo-elements Reference

:link	Unvisited <a> elements.
:visited	Visited <a> elements.
:hover	Element when mouse pointer is over it.
:active	Element being activated (clicked).
:focus	Element that has keyboard/mouse focus.
:focus-visible	Focus only when navigating by keyboard.
:checked	Checked checkbox or radio button.

:disabled	Disabled form element.
:required	Form field with required attribute.
:valid / :invalid	Form field validity state.
:first-child	Element that is the first child of its parent.
:last-child	Element that is the last child of its parent.
:nth-child(n)	Every nth child (supports formulas like 2n+1).
:not(selector)	Elements that do NOT match the selector.
:empty	Elements with no children.
::before	Insert generated content before element's content.
::after	Insert generated content after element's content.
::placeholder	Styles the placeholder text of an input.
::selection	Styles the portion of text selected by the user.
::first-line	Styles the first line of a block element.

JavaScript DOM Manipulation

The **Document Object Model (DOM)** is the browser's representation of the HTML document as a tree of objects. JavaScript can read and modify the DOM to make pages dynamic.

```
// Selecting elements
document.getElementById('myId');
document.querySelector('.myClass'); // first match
document.querySelectorAll('p.intro'); // NodeList
document.getElementsByTagName('div');

// Reading and writing content
el.textContent = 'New text'; // plain text
el.innerHTML = '<strong>Bold</strong>'; // HTML (XSS risk!)
el.getAttribute('href');
el.setAttribute('data-id', '42');
el.removeAttribute('disabled');

// CSS Classes
el.classList.add('active');
el.classList.remove('hidden');
el.classList.toggle('open');
el.classList.contains('active'); // boolean

// Inline styles
el.style.backgroundColor = '#1a3c6e';
```

```
el.style.display = 'none';

// Creating and inserting elements
const div = document.createElement('div');
div.textContent = 'Dynamic!';
div.className = 'card';
document.body.appendChild(div); // add at end
parent.insertBefore(div, existingChild); // add before
parent.removeChild(existingChild); // remove
el.replaceWith(newEl); // replace

// Traversal
el.parentElement;
el.children; // HTMLCollection of child elements
el.firstElementChild; el.lastElementChild;
el.nextElementSibling; el.previousElementSibling;
```

JavaScript Closures and Scope

A **closure** is a function that retains access to its outer function's variables even after the outer function has returned. Closures are fundamental to JavaScript and enable data privacy, factory functions, and event handlers.

```
// Basic closure
function makeCounter() {
let count = 0; // private variable
return {
increment() { count++; },
decrement() { count--; },
getCount() { return count; }
};
}
const counter = makeCounter();
counter.increment();
counter.increment();
counter.getCount(); // 2 (count is private, not accessible outside)

// Closure in event handlers
function attachHandler(element, message) {
element.addEventListener('click', () => {
alert(message); // 'message' is closed over
});
}

// Factory function using closure
function multiplier(factor) {
return (number) => number * factor;
}
const double = multiplier(2);
const triple = multiplier(3);
double(5); // 10
```

```
triple(5); // 15
```

TypeScript Advanced Types

```
// Intersection Types (combine multiple types)
type Employee = Person & { employeeId: number; department: string };

// Conditional Types
type IsString = T extends string ? 'yes' : 'no';
type Test = IsString; // 'yes'

// Mapped Types
type Readonly = { readonly [K in keyof T]: T[K] };
type Partial = { [K in keyof T]?: T[K] };
type Required = { [K in keyof T]-?: T[K] };

// Utility Types (built-in)
type PartialUser = Partial; // all props optional
type ReadonlyUser = Readonly; // all props readonly
type PickedUser = Pick; // select props
type OmitEmail = Omit; // exclude props
type RecordMap = Record; // { [k: string]: number }

// Template Literal Types
type EventName = 'click' | 'focus' | 'blur';
type HandlerName = `on${Capitalize}`;
// 'onClick' | 'onFocus' | 'onBlur'

// keyof and typeof
type UserKeys = keyof User; // 'id' | 'name' | 'email'
const config = { host: 'localhost', port: 3000 };
type Config = typeof config; // { host: string; port: number }
```

Servlet Filters and Listeners

Servlet Filters intercept requests and responses before they reach the servlet or after they leave it. They are used for logging, authentication, compression, encoding, etc.

```
// Implementing a Filter
@WebFilter('/*') // applies to all URLs
public class LoggingFilter implements Filter {

@Override
public void doFilter(ServletRequest request,
ServletResponse response,
FilterChain chain)
throws IOException, ServletException {

HttpServletRequest req = (HttpServletRequest) request;
long start = System.currentTimeMillis();
```

```
System.out.println('Incoming: ' + req.getRequestURI());

chain.doFilter(request, response); // continue to servlet

long elapsed = System.currentTimeMillis() - start;
System.out.println('Completed in ' + elapsed + 'ms');
}

@Override public void init(FilterConfig fc) {}
@Override public void destroy() {}
}

// Authentication Filter example
if (session == null || session.getAttribute('user') == null) {
response.sendRedirect('/login.html');
return; // don't continue filter chain
}
chain.doFilter(request, response);
```

Servlet Listeners

ServletContextListener	Notified when the web application starts or stops.
HttpSessionListener	Notified when a session is created or destroyed.
ServletRequestListener	Notified at the start and end of each request.
HttpSessionAttributeListener	Notified when session attributes are added/removed/replaced.
ServletContextAttributeListener	Notified for changes to ServletContext attributes.

JDBC Transactions

A **transaction** is a sequence of operations that must all succeed or all fail (atomicity). JDBC provides transaction control through the Connection object.

```
Connection conn = dataSource.getConnection();
conn.setAutoCommit(false); // disable auto-commit

try {
PreparedStatement debit = conn.prepareStatement(
'UPDATE accounts SET balance = balance - ? WHERE id = ?');
debit.setDouble(1, 500.00);
debit.setInt(2, fromAccountId);
debit.executeUpdate();

PreparedStatement credit = conn.prepareStatement(
'UPDATE accounts SET balance = balance + ? WHERE id = ?');
credit.setDouble(1, 500.00);
```

```
credit.setInt(2, toAccountId);
credit.executeUpdate();

conn.commit(); // both succeed: commit
System.out.println('Transfer successful');

} catch (SQLException e) {
conn.rollback(); // any failure: rollback both
System.err.println('Transfer failed, rolled back: ' + e.getMessage());
} finally {
conn.setAutoCommit(true);
conn.close();
}
```

Web Security Fundamentals

XSS (Cross-Site Scripting)	Attacker injects malicious scripts into web pages viewed by others. Prevent by escaping output and using Content-Security-Policy headers.
SQL Injection	Attacker manipulates SQL queries via user input. Prevent with PreparedStatement / parameterised queries.
CSRF (Cross-Site Request Forgery)	Tricks authenticated users into submitting unwanted requests. Prevent with CSRF tokens and SameSite cookie attribute.
Clickjacking	Embedding the target site in an invisible iframe. Prevent with X-Frame-Options or CSP frame-ancestors header.
Broken Authentication	Weak passwords, no account lockout, insecure session IDs. Use bcrypt for passwords, HTTPS-only cookies, session timeout.
Insecure Direct Object Reference	Accessing resources by guessing IDs. Validate ownership server-side on every request.
HTTPS / TLS	All production sites must use HTTPS. Obtain free certificates via Let's Encrypt.
CORS	Configure Cross-Origin Resource Sharing headers carefully to allow only trusted origins.

> *Security is not an afterthought – it must be built into every layer: HTML (output encoding), CSS (no data URIs from untrusted sources), JavaScript (sanitise DOM writes), and server-side (parameterised queries, authentication, authorisation).*