

## 1 Semantic Elements

### Definition:

HTML5 semantic elements clearly describe the meaning of the content inside them.

### Elements / Properties:

- <header> → Top section
- <footer> → Bottom section
- <nav> → Navigation links
- <section> → Grouped content
- <article> → Independent content
- <aside> → Sidebar
- <figure> → Image container
- <figcaption> → Caption for figure

### Example Description:

Structure a webpage to improve readability and accessibility.

### Example Code:

```
<header><h1>Website Title</h1></header>

<nav><a href="#">Home</a> | <a href="#">Blog</a></nav>

<section>

  <article><h2>Blog Post</h2><p>Article content</p></article>

  <aside><p>Sidebar content</p></aside>

</section>

<footer><p>&copy; 2025 MyWebsite</p></footer>
```

### Importance:

- ✓ Improves SEO
- ✓ Better accessibility
- ✓ Clean, maintainable code

---

## 2 HTML5 Multimedia

### Definition:

HTML5 allows embedding **video**, **audio**, and **subtitles** natively.

### Elements / Properties:

- <video> → controls, autoplay, loop, muted, width, height
- <audio> → controls, autoplay, loop, muted

- <track> → src, kind, srclang, label

**Example Description:**

Play videos and audio directly in webpages without plugins.

**Example Code:**

```
<video width="400" controls autoplay loop>
  <source src="movie.mp4" type="video/mp4">
</video>

<audio controls><source src="song.mp3" type="audio/mp3"></audio>

<video width="400" controls>
  <source src="movie.mp4" type="video/mp4">
  <track src="subtitles.vtt" kind="subtitles" srclang="en" label="English">
</video>
```

**Importance:**

- ✓ No plugins required
  - ✓ Works on all devices
  - ✓ Easy JS integration
- 

### 3 Canvas API

**Definition:**

<canvas> provides a **bitmap area** to draw graphics using JavaScript.

**Elements / Properties:**

- <canvas> → id, width, height, style
- JS Properties: getContext('2d'), fillStyle, fillRect(), arc(), beginPath(), stroke(), fill()

**Example Description:**

Used for dynamic graphics, games, charts, and animations.

**Example Code:**

```
<canvas id="myCanvas" width="400" height="300" style="border:1px solid black;"></canvas>

<script>

let c = document.getElementById("myCanvas");

let ctx = c.getContext("2d");

ctx.fillStyle = "blue";

ctx.fillRect(50,50,150,100);

ctx.beginPath();
```

```
ctx.arc(200,150,50,0,Math.PI*2);  
ctx.fillStyle = "red";  
ctx.fill();  
</script>
```

**Importance:**

- ✓ Dynamic graphics
  - ✓ Animations & games
  - ✓ Interactive applications
- 

## 4 SVG (Scalable Vector Graphics)

**Definition:**

SVG is **vector-based graphics** using XML, scalable and DOM manipulable.

**Elements / Properties:**

- <svg> → width, height, viewBox
- <circle> → cx, cy, r, fill, stroke
- <rect> → x, y, width, height, fill
- <line> → x1, y1, x2, y2
- <polyline> → points
- <text> → x, y, font-size, fill

**Example Description:**

Create icons, logos, and charts without losing quality.

**Example Code:**

**Importance:**

- ✓ Scalable
  - ✓ CSS & JS editable
  - ✓ Perfect for responsive design
- 

## 5 Web Storage API

**Definition:**

Web Storage allows **storing data in the browser** more securely than cookies.

**Elements / Properties:**

- localStorage → permanent storage

- sessionStorage → per session storage

#### **Example Description:**

Save user preferences or session data without server requests.

#### **Example Code:**

```
localStorage.setItem('name','John'); // permanent  
sessionStorage.setItem('name','John'); // session only
```

#### **Importance:**

- ✓ Larger storage capacity than cookies
  - ✓ No server requests
  - ✓ Persistent or session-based storage
- 

## **6 Web Workers**

#### **Definition:**

Web Workers run JavaScript in **background threads**, without blocking UI.

#### **Elements / Properties:**

- Worker() → create worker
- postMessage() → send data
- onmessage → receive data

#### **Example Description:**

Perform heavy calculations in background threads.

#### **Example Code:**

```
let worker = new Worker('worker.js');  
  
worker.onmessage = function(e) { console.log(e.data); };  
  
worker.postMessage('start');
```

#### **Importance:**

- ✓ Smooth UI
  - ✓ Handles large computations
  - ✓ Improves performance
- 

## **7 Geolocation API**

#### **Definition:**

Get user's **current location** (with permission).

#### **Elements / Properties:**

- navigator.geolocation.getCurrentPosition()

- coords.latitude, coords.longitude

**Example Description:**

Useful in maps, delivery apps, ride-booking apps.

**Example Code:**

```
navigator.geolocation.getCurrentPosition(  
    function(position) {  
        console.log(position.coords.latitude, position.coords.longitude);  
    }  
);
```

**Importance:**

- ✓ Location-based services
  - ✓ Improves user experience
  - ✓ Used in apps like maps & delivery
- 

## 8 HTML5 Form Enhancements

**Definition:**

HTML5 introduces new **input types and attributes** for validation.

**Elements / Properties:**

- Input types: email, number, url, date, range, color
- Attributes: required, placeholder, pattern, autocomplete, min, max, list

**Example Description:**

Enhances forms without JavaScript validation.

**Example Code:**

```
<input type="email" placeholder="Enter email" required pattern="[a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]{2,3}">  
<input type="date" min="2025-01-01" max="2025-12-31">
```

**Importance:**

- ✓ Built-in validation
  - ✓ Improves UX
  - ✓ Reduces JS usage
- 

## 9 Drag and Drop API

**Definition:**

Enable drag-and-drop functionality in browsers.

**Elements / Properties:**

- draggable="true"
- Events: dragstart, dragover, drop

**Example Description:**

Reorder elements or implement file uploads with drag-and-drop.

**Example Code:**

```
element.draggable = true;  
  
element.addEventListener('dragstart', function(e) {  
    e.dataTransfer.setData("text", e.target.id);  
});
```

**Importance:**

- ✓ Simplifies interactive UI
  - ✓ No heavy JS libraries needed
- 

## 10 WebSockets

**Definition:**

Two-way **real-time communication** between client and server.

**Elements / Properties:**

- new WebSocket(url)
- onopen, onmessage, onclose

**Example Description:**

Used for live chats, online games, or real-time dashboards.

**Example Code:**

```
let ws = new WebSocket("wss://example.com/socket");  
  
ws.onmessage = function(event) { console.log(event.data); };
```

**Importance:**

- ✓ Real-time interaction
- ✓ Low latency
- ✓ Perfect for interactive applications

## Advanced CSS3 Concepts

---

### **1 CSS Selectors (Advanced)**

#### **Definition:**

Selectors are used to **target HTML elements** for styling. Advanced selectors allow precise targeting.

#### **Properties / Types:**

- Attribute selectors → [type="text"]
- Pseudo-classes → :nth-child(), :not(), :focus-within
- Pseudo-elements → ::before, ::after, ::selection

#### **Example Description:**

Style specific elements without adding extra classes.

#### **Example Code:**

```
input[type="text"] { border: 1px solid blue; }

p:nth-child(2) { color: green; }

p::before { content: "Note: "; font-weight: bold; }
```

#### **Importance:**

- ✓ Precise styling
  - ✓ Reduces extra classes
  - ✓ Cleaner CSS
- 

### **2 Flexbox Layout**

#### **Definition:**

A one-dimensional layout system for arranging items **in a row or column**.

#### **Properties:**

- display: flex → enable flex container
- justify-content → horizontal alignment
- align-items → vertical alignment
- flex-direction → row / column / reverse
- flex-wrap → wrap items

#### **Example Description:**

Align cards, navigation items, or layout sections easily.

#### **Example Code:**

```
.container {
```

```
display: flex;  
justify-content: center;  
align-items: center;  
flex-direction: row;  
flex-wrap: wrap;  
}
```

**Importance:**

- ✓ Simplifies alignment
  - ✓ Responsive-friendly
  - ✓ Great for UI components
- 

### 3 CSS Grid Layout

**Definition:**

A two-dimensional layout system using **rows** and **columns**.

**Properties:**

- display: grid → enable grid container
- grid-template-rows → define row heights
- grid-template-columns → define column widths
- grid-gap → spacing between items
- grid-area → assign items to grid areas

**Example Description:**

Ideal for dashboards, galleries, and complex layouts.

**Example Code:**

```
.grid {  
  
display: grid;  
  
grid-template-rows: 100px 200px;  
  
grid-template-columns: 1fr 2fr;  
  
grid-gap: 10px;  
}
```

**Importance:**

- ✓ Complex layouts without floats
- ✓ Responsive-friendly
- ✓ Powerful for structured designs

---

## 4 CSS Transitions & Animations

### Definition:

- **Transitions** → smooth changes of properties
- **Animations** → keyframe-based property changes

### Properties:

- transition: property duration timing-function
- @keyframes → define animation steps
- animation-name, animation-duration

### Example Description:

Animate buttons, hover effects, or fading elements.

### Example Code:

```
div { transition: all 0.5s ease; }
```

```
@keyframes fadeIn {  
  from { opacity: 0; }  
  to { opacity: 1; }  
}
```

```
div { animation: fadeIn 2s; }
```

### Importance:

- ✓ Smooth UX
- ✓ Eye-catching designs
- ✓ Highlights interactivity

---

## 5 CSS Variables (Custom Properties)

### Definition:

Reusable CSS variables for colors, fonts, sizes, etc.

### Properties:

- :root { --variable-name: value; }
- var(--variable-name) → use variable

### Example Description:

Easily change themes or reuse values.

### **Example Code:**

```
:root { --main-color: #ff6600; }

h1 { color: var(--main-color); }
```

### **Importance:**

- ✓ DRY principle (Don't Repeat Yourself)
  - ✓ Dynamic theming
  - ✓ Easy maintainability
- 

## **6 Responsive Web Design (RWD)**

### **Definition:**

Design webpages that **adapt to screen sizes**.

### **Properties / Techniques:**

- Media Queries → @media (max-width: 768px)
- Flexible layouts → %, fr, vw/vh
- Responsive images → <img srcset="">, <picture>

### **Example Code:**

```
@media (max-width:768px) {

  body { font-size: 14px; }

}
```

### **Importance:**

- ✓ Works on mobile, tablet, desktop
  - ✓ Better user experience
  - ✓ Required for modern web
- 

## **7 CSS Preprocessing Concepts**

### **Definition:**

Preprocessing features help **write advanced CSS** more efficiently.

### **Properties / Concepts:**

- Nesting
- Mixins
- Variables
- Partials
- Functions

### **Example Description:**

Organize large CSS code for maintainability.

### **Example Code (SCSS Example):**

```
$main-color: #ff6600;  
  
.card {  
  color: $main-color;  
  
.title { font-weight: bold; }  
  
}
```

### **Importance:**

- ✓ Modular CSS
  - ✓ Reusable code
  - ✓ Easier maintenance
- 

## Transformations

### **Definition:**

Move, rotate, scale, or skew elements in 2D or 3D space.

### **Properties:**

- transform: rotate() scale() translate() skew() perspective()

### **Example Code:**

```
div { transform: rotate(45deg) scale(1.2) translate(10px,20px); }
```

### **Importance:**

- ✓ Enhances UI
  - ✓ Creates dynamic effects
  - ✓ Interactive designs
- 

## CSS Filters

### **Definition:**

Apply **visual effects** like blur, brightness, grayscale to elements.

### **Properties:**

- filter: blur(px) brightness(%) grayscale(%) contrast(%)

### **Example Code:**

```
img { filter: blur(2px) grayscale(50%) contrast(120%); }
```

### **Importance:**

- ✓ Image effects without Photoshop

- ✓ Enhances UI/UX
  - ✓ Interactive hover effects
- 

## 10 CSS Specificity & Inheritance

### Definition:

Defines which styles take precedence.

### Rules / Properties:

- Inline > ID > Class > Element
- !important overrides everything (use carefully)

### Example Code:

```
p { color: blue; }

#para { color: red; } /* Higher specificity */
```

### Importance:

- ✓ Controls style application
  - ✓ Avoids CSS conflicts
- 

## 11 BEM Methodology

### Definition:

BEM = Block, Element, Modifier → **modular CSS naming**.

### Example Code:

```
.card__title--active { color: red; }
```

### Importance:

- ✓ Keeps CSS modular
  - ✓ Easy maintenance
  - ✓ Scalable for large projects
- 

## 12 Advanced Media Queries

### Definition:

Target **user preferences, devices, and orientations**.

### Properties / Examples:

- prefers-color-scheme: dark → dark mode
- prefers-reduced-motion → reduce animations
- Orientation → portrait / landscape

- Aspect ratio → aspect-ratio: 16/9

**Example Code:**

```
@media (prefers-color-scheme: dark) {  
  body { background: #000; color: #fff; }  
}
```

**Importance:**

- ✓ Responsive & accessible
  - ✓ Improves user experience
  - ✓ Adaptive to devices
- 

## **1 3 CSS Clipping & Masking**

**Definition:**

Create **shapes and cutout effects** for elements.

**Properties / Examples:**

- clip-path: circle(50% at 50% 50%)
- mask-image: url(mask.png)

**Example Code:**

```
div { clip-path: circle(50% at 50% 50%); }  
  
img { mask-image: url(mask.png); }
```

**Importance:**

- ✓ Creative UI designs
- ✓ Dynamic shapes without images
- ✓ Enhances aesthetics

## JavaScript ES6+ Modern Features

---

### 1 let and const

#### Definition:

let and const are **block-scoped variable declarations**, replacing var in modern JS.

#### Example Code:

```
let name = "Alice"; // can be reassigned  
const PI = 3.14; // cannot be reassigned
```

#### Importance:

- ✓ Avoids variable hoisting issues
  - ✓ Prevents accidental reassignment (const)
  - ✓ Safer and more readable code
- 

### 2 Arrow Functions

#### Definition:

Shorter syntax for functions, with lexical this.

#### Example Code:

```
const add = (a, b) => a + b;  
const greet = name => `Hello, ${name}!`;
```

#### Importance:

- ✓ Cleaner syntax
  - ✓ No binding issues with this
  - ✓ Great for callbacks and functional programming
- 

### 3 Template Literals

#### Definition:

String literals using backticks ` with **interpolation** and multi-line support.

#### Example Code:

```
let name = "Bob";  
  
let message = `Hello, ${name}!  
Welcome to JavaScript ES6+!`;
```

#### Importance:

- ✓ Easier string concatenation

- ✓ Multi-line strings without \n
  - ✓ Cleaner and readable code
- 

## 4 Default Parameters

### Definition:

Function parameters can have default values.

### Example Code:

```
function greet(name = "Guest") {  
  console.log(`Hello, ${name}!`);  
}  
  
greet();      // Hello, Guest!  
  
greet("Alice"); // Hello, Alice!
```

### Importance:

- ✓ Simplifies function handling
  - ✓ Avoids undefined parameters
- 

## 5 Destructuring Assignment

### Definition:

Extract values from arrays or objects into variables.

### Example Code:

```
// Array  
  
const colors = ["red", "green", "blue"];  
  
const [first, second] = colors;
```

### // Object

```
const person = { name: "Alice", age: 25 };  
  
const { name, age } = person;
```

### Importance:

- ✓ Cleaner code
  - ✓ Avoids repetitive assignments
  - ✓ Works with nested objects
- 

## 6 Spread & Rest Operator

**Definition:**

- **Spread (...)** → expand arrays/objects
- **Rest (...)** → collect remaining elements

**Example Code:**

```
// Spread  
const arr1 = [1,2];  
const arr2 = [...arr1, 3,4];
```

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

**Importance:**

- ✓ Easier array/object manipulation
  - ✓ Flexible function parameters
- 

## 7 Enhanced Object Literals

**Definition:**

Simpler syntax for creating objects.

**Example Code:**

```
let name = "Alice";  
let age = 25;  
const person = { name, age, greet() { console.log("Hi"); } };
```

**Importance:**

- ✓ Cleaner, shorter code
  - ✓ Dynamic property names
- 

## 8 Modules (import/export)

**Definition:**

Enable modular code by **importing/exporting** functions, objects, or variables.

**Example Code:**

```
// utils.js  
export function add(a,b){ return a+b; }
```

```
// app.js

import { add } from './utils.js';

console.log(add(2,3));
```

**Importance:**

- ✓ Reusable code
  - ✓ Better project organization
  - ✓ Supports modern JS frameworks
- 

## 9 Promises

**Definition:**

Handle asynchronous operations with `then/catch` instead of callbacks.

**Example Code:**

```
let promise = new Promise((resolve, reject) => {

  let success = true;

  success ? resolve("Done") : reject("Error");

});
```

```
promise.then(msg => console.log(msg))

  .catch(err => console.error(err));
```

**Importance:**

- ✓ Avoids callback hell
  - ✓ Cleaner async handling
  - ✓ Works with `async/await`
- 

## 10 Async/Await

**Definition:**

Syntactic sugar over Promises for **synchronous-like async code**.

**Example Code:**

```
async function fetchData() {

  try {

    let response = await fetch('https://api.example.com/data');

    let data = await response.json();
```

```
    console.log(data);
} catch(err) {
    console.error(err);
}
}

fetchData();
```

**Importance:**

- ✓ Easier to read and write async code
  - ✓ Reduces nested then() chains
- 

## 1 1 Classes

**Definition:**

Syntactic sugar for constructor functions and prototype-based inheritance.

**Example Code:**

```
class Person {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }
    greet() { console.log(`Hello, ${this.name}`); }
}
```

```
const alice = new Person("Alice", 25);
```

```
alice.greet();
```

**Importance:**

- ✓ Cleaner OOP in JS
  - ✓ Readable and maintainable code
- 

## 1 2 Default + Named Exports

**Definition:**

Modules can have one default export and multiple named exports.

**Example Code:**

```
// utils.js
```

```
export const add = (a,b) => a+b;  
export default function greet(){ console.log("Hello"); }
```

```
// app.js  
  
import greet, { add } from './utils.js';  
  
greet(); // Hello  
  
console.log(add(2,3));
```

**Importance:**

- ✓ Flexible module system
  - ✓ Cleaner imports
- 

## 1 3 Optional Chaining (?)

**Definition:**

Access deeply nested object properties safely.

**Example Code:**

```
let user = { profile: { name: "Alice" } };  
  
console.log(user.profile?.name); // Alice  
  
console.log(user.address?.city); // undefined, no error
```

**Importance:**

- ✓ Prevents runtime errors
  - ✓ Cleaner code
- 

## 1 4 Nullish Coalescing (??)

**Definition:**

Provide a default value only if **null** or **undefined**.

**Example Code:**

```
let name = null;  
  
let user = name ?? "Guest";  
  
console.log(user); // Guest
```

**Importance:**

- ✓ Safer than ||
  - ✓ Avoids falsey value issues
-

## 1 5 Map & Set

### Definition:

- **Map** → key-value collection (any type as key)
- **Set** → unique collection of values

### Example Code:

```
let map = new Map();  
  
map.set('a', 1);  
  
map.set('b', 2);  
  
  
let set = new Set([1,2,3,2]);  
  
console.log([...set]); // [1,2,3]
```

### Importance:

- ✓ Efficient data structures
  - ✓ Better than plain objects/arrays for certain operations
- 

I can also create a **complete PDF notes version for ES6+ features** like we did for HTML5 & CSS3, with **code, examples, and diagrams**.

Do you want me to make that PDF too?

### You said:

JavaScript ES6+ modern features i need every concept in this

### ChatGPT said:

Absolutely! Let's create a **complete, detailed, beginner-friendly guide to all modern JavaScript ES6+ features**. I'll include **definition, syntax, example code, properties/methods, and why it's important** for each concept.

---

## JavaScript ES6+ Modern Features – Complete Notes

---

### 1. let and const

**Definition:** Block-scoped variables, safer than var.

### Properties / Features:

- **let** → can be reassigned, block-scoped
- **const** → cannot be reassigned, block-scoped

### Example Code:

```
let count = 1;  
count = 2; // ✅ works
```

```
const PI = 3.14;  
PI = 3.1415; // ❌ error
```

**Importance:** Prevents accidental reassessments, avoids hoisting issues, cleaner scope handling.

---

## 2. Arrow Functions

**Definition:** Concise function syntax with lexical this.

**Properties / Features:**

- Implicit return for single expressions
- Lexical this binding

**Example Code:**

```
const sum = (a, b) => a + b;  
const greet = name => `Hello, ${name}!`;
```

**Importance:** Cleaner syntax, easier callbacks, avoids this issues.

---

## 3. Template Literals

**Definition:** Multi-line strings and string interpolation using backticks `.

**Properties / Features:**

- \${expression} syntax
- Multi-line strings supported

**Example Code:**

```
let name = "Alice";  
let msg = `Hello, ${name}!  
Welcome to ES6+ JavaScript!`;
```

**Importance:** Simplifies string operations, improves readability.

---

## 4. Default Parameters

**Definition:** Assign default values to function parameters.

**Properties / Features:**

- Avoids undefined for missing arguments

**Example Code:**

```
function greet(name = "Guest") {  
    console.log(`Hello, ${name}!`);  
}  
  
greet(); // Hello, Guest!  
  
greet("Bob"); // Hello, Bob!
```

**Importance:** Cleaner functions, prevents undefined arguments.

---

## 5. Destructuring Assignment

**Definition:** Extract values from arrays or objects.

**Properties / Features:**

- Works with nested structures
- Can rename variables during destructuring

**Example Code:**

```
// Array  
  
const colors = ["red", "green", "blue"];  
  
const [first, second] = colors;
```

```
// Object
```

```
const person = { name: "Alice", age: 25 };  
  
const { name, age } = person;
```

**Importance:** Reduces repetitive code, cleaner variable extraction.

---

## 6. Spread & Rest Operators

**Definition:** ... operator to expand (spread) or gather (rest) elements.

**Properties / Features:**

- Spread → copy arrays/objects, pass as arguments
- Rest → collect remaining elements

**Example Code:**

```
// Spread  
  
const arr1 = [1,2];  
  
const arr2 = [...arr1, 3, 4];
```

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

**Importance:** Simplifies array/object manipulation and flexible functions.

---

## 7. Enhanced Object Literals

**Definition:** Shorthand syntax for objects and dynamic property names.

**Properties / Features:**

- Property shorthand, method shorthand, computed keys
- Example Code:**

```
let name = "Alice", age = 25;  
  
const person = {  
    name,  
    age,  
    greet() { console.log("Hi"); }  
};
```

**Importance:** Cleaner, concise code, dynamic object creation.

---

## 8. Modules (import/export)

**Definition:** Organize code into reusable modules.

**Properties / Features:**

- export → named or default
- import → bring modules into other files

**Example Code:**

```
// utils.js  
  
export function add(a,b){ return a+b; }
```

```
// app.js  
  
import { add } from './utils.js';  
  
console.log(add(2,3));
```

**Importance:** Code modularity, better maintenance, reusable components.

---

## 9. Promises

**Definition:** Handle asynchronous operations.

**Properties / Features:**

- States: pending, fulfilled, rejected
- then(), catch() for handling results

**Example Code:**

```
let promise = new Promise((resolve,reject) => {  
    let success = true;  
    success ? resolve("Done") : reject("Error");  
});  
promise.then(msg => console.log(msg))  
    .catch(err => console.error(err));
```

**Importance:** Avoids callback hell, cleaner async handling.

---

## 10 Async/Await

**Definition:** Synchronous-style async code over Promises.

**Properties / Features:**

- Use with async functions
- await waits for Promise resolution

**Example Code:**

```
async function fetchData() {  
    try {  
        let response = await fetch('https://api.example.com/data');  
        let data = await response.json();  
        console.log(data);  
    } catch(err) {  
        console.error(err);  
    }  
}  
fetchData();
```

**Importance:** Readable asynchronous code, avoids nested .then() chains.

---

## 1 1 Classes

**Definition:** Modern OOP in JS, syntactic sugar over prototypes.

**Properties / Features:**

- constructor() for initialization
- extends for inheritance
- Methods defined inside class

**Example Code:**

```
class Person {  
  constructor(name, age) { this.name = name; this.age = age; }  
  greet() { console.log(`Hello, ${this.name}`); }  
}  
  
const alice = new Person("Alice", 25);  
alice.greet();
```

**Importance:** Cleaner object-oriented code, easier inheritance.

---

## 1 2 Default + Named Exports

**Definition:** Modules can have one default and multiple named exports.

**Example Code:**

```
// utils.js  
  
export const add = (a,b) => a+b;  
export default function greet(){ console.log("Hello"); }
```

```
// app.js  
  
import greet, { add } from './utils.js';  
greet(); // Hello  
console.log(add(2,3));
```

**Importance:** Flexible modularization.

---

## 1 3 Optional Chaining (?.)

**Definition:** Safely access nested object properties.

**Example Code:**

```
let user = { profile: { name: "Alice" } };
console.log(user.profile?.name); // Alice
console.log(user.address?.city); // undefined
```

**Importance:** Prevents runtime errors, cleaner code.

---

## 1 4 Nullish Coalescing (??)

**Definition:** Provide default value if null/undefined only.

**Example Code:**

```
let name = null;
let user = name ?? "Guest";
console.log(user); // Guest
```

**Importance:** Avoids falsey value issues with ||.

---

## 1 5 Map & Set

**Definition:** New ES6 data structures.

**Properties / Features:**

- Map → key-value, any key type
- Set → unique values

**Example Code:**

```
let map = new Map();
map.set('a',1); map.set('b',2);
```

```
let set = new Set([1,2,2,3]);
console.log([...set]); // [1,2,3]
```

**Importance:** Efficient data management, better than objects/arrays for certain use-cases.

---

## 1 6 Symbols

**Definition:** Unique and immutable primitive values.

**Example Code:**

```
const sym1 = Symbol("id");
const sym2 = Symbol("id");
console.log(sym1 === sym2); // false
```

**Importance:** Used for private object properties, unique keys.

---

## 1 7 For...of Loop

**Definition:** Iterates over iterable objects (arrays, strings, maps, sets).

**Example Code:**

```
let arr = [1,2,3];  
for(let val of arr) { console.log(val); }
```

**Importance:** Cleaner iteration over arrays than forEach.

---

## 1 8 Default & Named Parameters in Functions

Already covered in **Default Parameters** and **Modules**, but often combined for clarity.

---

## 1 9 BigInt

**Definition:** Handle integers beyond Number.MAX\_SAFE\_INTEGER.

**Example Code:**

```
const bigNum = 123456789012345678901234567890n;  
console.log(bigNum + 1n);
```

**Importance:** Handles very large numbers safely.

---

## 2 0 WeakMap & WeakSet

**Definition:** Maps/Sets with weakly held object keys (garbage collectible).

**Example Code:**

```
let wm = new WeakMap();  
  
let obj = {};  
wm.set(obj, "value");
```

**Importance:** Memory-efficient, used in advanced JS patterns.

## 1 What is React.js?

### Definition:

React.js is a **JavaScript library** for building **user interfaces**, especially **single-page applications (SPAs)**.

- Maintained by Facebook
- Component-based architecture

### Example Code:

```
import React from 'react';
import ReactDOM from 'react-dom';
```

```
function App() {
  return <h1>Hello, React!</h1>;
}
```

```
ReactDOM.render(<App />, document.getElementById('root'));
```

### Importance:

- ✓ Component-based → reusable UI
  - ✓ Efficient → uses Virtual DOM
  - ✓ Declarative → easier to manage UI states
- 

## 2 JSX (JavaScript XML)

### Definition:

JSX is a **syntax extension of JavaScript** that allows writing HTML-like code in JS.

### Example Code:

```
const element = <h1>Hello, JSX!</h1>;
```

### Properties / Rules:

- Must return a single root element
- Can embed expressions: {expression}
- Class attribute → className
- HTML attributes follow camelCase (onClick, tabIndex)

### Importance:

- ✓ Cleaner code

- ✓ Declarative UI
  - ✓ Integrates JS logic into HTML structure
- 

### 3 Components

**Definition:** Components are **reusable building blocks** in React.

- **Functional Components** → function-based
- **Class Components** → ES6 classes

**Example Code (Functional):**

```
function Greeting(props) {  
  return <h1>Hello, {props.name}!</h1>;  
}
```

**Example Code (Class):**

```
class Greeting extends React.Component {  
  render() {  
    return <h1>Hello, {this.props.name}!</h1>;  
  }  
}
```

**Importance:**

- ✓ Modular, reusable code
  - ✓ Separates UI into manageable pieces
- 

### 4 Props (Properties)

**Definition:** Props are **read-only values** passed from parent to child components.

**Example Code:**

```
function Child(props) {  
  return <p>{props.message}</p>;  
}
```

```
function Parent() {  
  return <Child message="Hello from Parent!" />;  
}
```

**Importance:**

- ✓ Enables component communication
  - ✓ Reusability with dynamic data
- 

**5 State**

**Definition:** State is a **component's local data** that can change over time.

**Example Code (Functional + Hook):**

```
import React, { useState } from 'react';
```

```
function Counter() {  
  const [count, setCount] = useState(0);  
  
  return (  
    <div>  
      <p>Count: {count}</p>  
      <button onClick={() => setCount(count + 1)}>Increment</button>  
    </div>  
  );  
}
```

**Importance:**

- ✓ Controls dynamic UI
  - ✓ Tracks user interactions
- 

**6 Event Handling**

**Definition:** React uses **camelCase syntax** for events.

**Example Code:**

```
function Button() {  
  function handleClick() {  
    alert('Button clicked!');  
  }  
  
  return <button onClick={handleClick}>Click Me</button>;  
}
```

**Importance:**

- ✓ Interactive UI
  - ✓ Integrates seamlessly with state updates
- 

**7 Conditional Rendering**

**Definition:** Render elements **based on conditions**.

**Example Code:**

```
function Greeting({ isLoggedIn }) {  
  return <h1>{isLoggedIn ? "Welcome back!" : "Please sign in"}</h1>;  
}
```

**Importance:**

- ✓ Dynamic UI based on app state
- 

**8 Lists and Keys**

**Definition:** Rendering **arrays of elements** using map() with unique key props.

**Example Code:**

```
const items = ['Apple', 'Banana', 'Orange'];
```

```
function ItemList() {  
  return (  
    <ul>  
      {items.map((item, index) => (  
        <li key={index}>{item}</li>  
      ))}  
    </ul>  
  );  
}
```

**Importance:**

- ✓ Efficient updates
  - ✓ Prevents React warnings
  - ✓ Optimized rendering
-

## 9 Forms

**Definition:** Controlled components manage form inputs with state.

**Example Code:**

```
function NameForm() {  
  const [name, setName] = useState("");  
  
  const handleChange = (e) => setName(e.target.value);  
  const handleSubmit = (e) => {  
    e.preventDefault();  
    alert(`Submitted: ${name}`);  
  };  
  
  return (  
    <form onSubmit={handleSubmit}>  
      <input type="text" value={name} onChange={handleChange} />  
      <button type="submit">Submit</button>  
    </form>  
  );  
}
```

**Importance:**

- ✓ React manages form data
  - ✓ Real-time validation and dynamic forms
- 

## 10 React Hooks (Functional Components)

Hooks are **special functions** that let you use state and lifecycle features in functional components.

---

### useState

**Definition:** Add state to functional components.

**Example Code:**

```
const [count, setCount] = useState(0);
```

---

### useEffect

**Definition:** Handle side-effects (fetching data, subscriptions, DOM updates).

**Example Code:**

```
useEffect(() => {  
  console.log('Component mounted or updated');  
}, [count]); // runs when count changes
```

---

### useContext

**Definition:** Access global context data without prop drilling.

**Example Code:**

```
const ThemeContext = React.createContext('light');
```

```
function DisplayTheme() {  
  const theme = useContext(ThemeContext);  
  return <p>Theme: {theme}</p>;  
}
```

---

### useReducer

**Definition:** Alternative to useState for complex state logic.

**Example Code:**

```
const initialState = 0;  
  
function reducer(state, action) {  
  switch(action) {  
    case 'increment': return state + 1;  
    case 'decrement': return state - 1;  
    default: return state;  
  }  
}  
  
const [count, dispatch] = useReducer(reducer, initialState);
```

---

### useRef

**Definition:** Access DOM nodes or store mutable variables.

**Example Code:**

```
const inputRef = useRef(null);  
<input ref={inputRef} />;  
inputRef.current.focus();
```

---

## useMemo & useCallback

**Definition:** Optimize performance.

- useMemo → memoizes computed values
- useCallback → memoizes functions

**Example Code:**

```
const memoizedValue = useMemo(() => computeExpensiveValue(a,b), [a,b]);  
const memoizedCallback = useCallback(() => doSomething(a), [a]);
```

---

## Importance of Hooks

- ✓ State management in functional components
  - ✓ Lifecycle handling without class components
  - ✓ Cleaner and reusable logic
- 

## 1 1 React Router (Optional for SPA Navigation)

**Definition:** Declarative routing for React SPAs.

**Example Code:**

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
```

```
function App() {  
  return (  
    <Router>  
      <Routes>  
        <Route path="/" element={<Home />} />  
        <Route path="/about" element={<About />} />  
      </Routes>  
    </Router>  
  );  
}
```

**Importance:**

- ✓ Enables multi-page feel in SPAs
- ✓ URL-based navigation

## React.js State Management – Basics

---

### 1 What is State in React?

#### Definition:

State is a **JavaScript object** that holds data that may **change over time** in a component.

- State is **local to a component**
- Updating state causes **re-rendering of the component**

#### Example:

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

#### Importance:

- ✓ Makes UI dynamic
- ✓ Tracks user interactions
- ✓ Ensures component reactivity

---

### 2 Local State

#### Definition:

State that is **specific to a single component**. Managed using useState hook (functional components) or this.state (class components).

#### Functional Component Example:

```
function TodoItem() {
  const [done, setDone] = useState(false);
```

```

    return (
      <div>
        <p>{done ? "Completed" : "Pending"}</p>
        <button onClick={() => setDone(!done)}>Toggle</button>
      </div>
    );
}

```

**Importance:**

- ✓ Encapsulated within the component
  - ✓ Easy to manage small, independent pieces of state
- 

### 3 Shared / Global State

**Definition:**

State shared between **multiple components**.

- Managed via **lifting state up** or **state management libraries**

**Example – Lifting State Up:**

```

function Parent() {
  const [name, setName] = useState("Alice");
  return (
    <>
    <Child1 name={name} />
    <Child2 setName={setName} />
    </>
  );
}

function Child1({ name }) { return <p>Name: {name}</p>; }

function Child2({ setName }) {
  return <button onClick={() => setName("Bob")}>Change Name</button>;
}

```

### **Importance:**

- ✓ Allows multiple components to sync data
  - ✓ Avoids prop-drilling complexity with libraries
- 

## **Derived State**

### **Definition:**

State that can be **computed from props or other state** rather than stored directly.

### **Example:**

```
function ShoppingCart({ items }) {  
  const total = items.reduce((sum, item) => sum + item.price, 0); // derived  
  return <p>Total Price: ${total}</p>;  
}
```

### **Importance:**

- ✓ Reduces unnecessary state
  - ✓ Improves performance and readability
- 

## **State in Class Components**

### **Definition:**

Class components use `this.state` and `this.setState` to manage state.

### **Example:**

```
class Counter extends React.Component {  
  constructor() {  
    super();  
    this.state = { count: 0 };  
  }  
  increment = () => { this.setState({ count: this.state.count + 1 }); }  
  render() {  
    return (  
      <div>  
        <p>Count: {this.state.count}</p>  
        <button onClick={this.increment}>Increment</button>  
      </div>  
    );  
  }  
}
```

```
    }  
}  
}
```

**Importance:**

- ✓ Supports lifecycle methods
  - ✓ Still used in older React code
- 

## 6 State Management Libraries

For large applications, managing state only with useState can be cumbersome. Popular libraries:

**a) Redux**

- Centralized store
- Actions and reducers update state
- Works well with large, complex apps

**Basic Example:**

```
const initialState = { count: 0 };
```

```
function reducer(state = initialState, action) {  
  switch(action.type){  
    case 'INCREMENT': return { count: state.count + 1 };  
    default: return state;  
  }  
}
```

**b) Context API**

- Built-in React solution
- Provides **global state** without prop drilling

**Example:**

```
const ThemeContext = React.createContext('light');
```

```
function App() {  
  return (  
    <ThemeContext.Provider value="dark">  
      <Child />
```

```
</ThemeContext.Provider>
);

}

function Child() {
  const theme = React.useContext(ThemeContext);
  return <p>Theme: {theme}</p>;
}
```

#### **Importance of State Libraries:**

- ✓ Avoid prop drilling
  - ✓ Easier to maintain global data
  - ✓ Makes complex apps scalable
- 

#### **7 Best Practices**

- Keep state as **local as possible**
  - Use **derived state** instead of storing computed values
  - Avoid **deeply nested state**; use reducers or libraries
  - Use **hooks** in functional components (`useState`, `useReducer`)
- 

#### **Key Takeaways**

- State = dynamic data of a component
- `useState` for local functional component state
- Lift state up for sharing between components
- Use Context or Redux for large-scale global state

## CSS Frameworks

### Definition:

CSS frameworks are **pre-prepared libraries** that provide **ready-to-use CSS rules, components, and layouts**, helping developers **build responsive and consistent UI quickly**.

---

### Popular CSS Frameworks

#### a) Bootstrap

- Most popular, mobile-first framework
- Uses **grid system** for layout
- Prebuilt **components**: buttons, forms, modals, navbar

#### Example Code:

```
<link rel="stylesheet"  
      href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
```

```
<div class="container">  
  <div class="row">  
    <div class="col-md-6">Column 1</div>  
    <div class="col-md-6">Column 2</div>  
  </div>  
  <button class="btn btn-primary">Click Me</button>  
</div>
```

#### Importance:

- ✓ Rapid UI development
  - ✓ Responsive out-of-the-box
  - ✓ Consistent cross-browser design
- 

#### b) Tailwind CSS

- Utility-first CSS framework
- Write **classes directly in HTML** for styling

#### Example Code:

```
<div class="bg-blue-500 text-white p-4 rounded">
```

Hello Tailwind!

```
</div>
```

**Importance:**

- ✓ Highly customizable
  - ✓ No unused CSS → smaller file size
  - ✓ Great for modern front-end projects
- 

**c) Materialize CSS**

- Based on Google Material Design
- Components with **animations, cards, icons**

**Example Code:**

```
<div class="card">  
  <div class="card-content">  
    <span class="card-title">Card Title</span>  
    <p>This is a Materialize card.</p>  
  </div>  
</div>
```

**Importance:**

- ✓ Consistent Material Design UI
  - ✓ Ready-to-use components
- 

**d) Bulma**

- Modern CSS framework
- Flexbox-based, lightweight
- Easy to learn

**Example Code:**

```
<div class="columns">  
  <div class="column is-half">Column 1</div>  
  <div class="column is-half">Column 2</div>  
</div>
```

**Importance:**

- ✓ Lightweight and easy

- ✓ Flexbox grid system
  - ✓ No JavaScript dependency
- 

## 2 CSS Preprocessors

### Definition:

CSS preprocessors are **languages that extend CSS** with extra features like **variables, nesting, mixins, and functions**, then **compile to standard CSS**.

---

### Popular CSS Preprocessors

#### a) SASS / SCSS

- Most popular preprocessor
- Supports **variables, nesting, mixins, inheritance**

#### Example Code:

```
$primary-color: #3498db;

body {
    font-family: Arial, sans-serif;
    background-color: $primary-color;

    .button {
        padding: 10px 20px;
        border-radius: 5px;
        &:hover {
            background-color: darken($primary-color, 10%);
        }
    }
}
```

#### Compiled CSS:

```
body {
    font-family: Arial, sans-serif;
    background-color: #3498db;
}
```

```
body .button {  
    padding: 10px 20px;  
    border-radius: 5px;  
}  
  
body .button:hover {  
    background-color: #2c80b4;  
}
```

**Importance:**

- ✓ Modular, maintainable CSS
  - ✓ Avoids repetitive code
  - ✓ Large projects easier to manage
- 

**b) LESS**

- Similar to SASS
- Uses **variables, mixins, nested rules**

**Example Code:**

```
@primary-color: #e74c3c;
```

```
.container {  
    color: @primary-color;  
}  
  
.button {  
    padding: 10px;  
    border-radius: 5px;  
}
```

**Importance:**

- ✓ Cleaner, organized CSS
  - ✓ Faster development
- 

**c) Stylus**

- Flexible preprocessor, supports concise syntax
- Powerful functions and mixins

### Example Code:

```
primary-color = #2ecc71
```

```
body  
background-color primary-color  
font-family Arial, sans-serif
```

### Importance:

- ✓ Flexible syntax
  - ✓ Advanced features for complex UI
- 

## 3 Benefits of Frameworks & Preprocessors

Feature	Framework	Preprocessor
Reusability	<input checked="" type="checkbox"/> Components & utilities	<input checked="" type="checkbox"/> Mixins, variables
Faster Development	<input checked="" type="checkbox"/> Prebuilt CSS	<input checked="" type="checkbox"/> Modular code
Consistency	<input checked="" type="checkbox"/> Standardized design	<input checked="" type="checkbox"/> Avoid repeated styles
Responsiveness	<input checked="" type="checkbox"/> Grid & helpers	<input checked="" type="checkbox"/> Needs manual media queries
Maintainability	<input checked="" type="checkbox"/> Easy to read HTML classes	<input checked="" type="checkbox"/> Easier large-scale CSS management

---

## 4 Best Practices

1. Use **frameworks** for rapid prototyping and responsive design.
  2. Use  **preprocessors** for **large projects** needing reusable and maintainable CSS.
  3. Combine Tailwind / Bootstrap with SCSS for **flexibility + maintainability**.
  4. Keep your **CSS modular** and organized.
- 

## ✓ Key Takeaways

- CSS frameworks → ready-made design & responsive layouts
- Preprocessors → enhance CSS with **variables, nesting, and mixins**
- Use them to **save time, reduce redundancy, and improve maintainability**

## Web Accessibility Standards – Complete Notes

---

### **1 What is Web Accessibility?**

#### **Definition:**

Web accessibility ensures that **websites, apps, and digital content are usable by everyone**, including people with **disabilities** such as visual, auditory, motor, or cognitive impairments.

#### **Importance:**

- ✓ Promotes inclusivity
  - ✓ Complies with legal regulations (ADA, WCAG)
  - ✓ Improves usability for all users
  - ✓ SEO-friendly
- 

### **2 Key Standards & Guidelines**

#### **a) WCAG (Web Content Accessibility Guidelines)**

- Published by **W3C (World Wide Web Consortium)**
- Defines how to make web content accessible
- Organized by **four principles**: POUR

#### **POUR Principles:**

Principle	Description
<b>Perceivable</b>	Information must be presented in ways users can perceive (text alternatives for images, captions for videos).
<b>Operable</b>	Users can navigate and use interface (keyboard accessible, clear navigation).
<b>Understandable</b>	Content and UI should be clear and predictable.
<b>Robust</b>	Content works across different devices, browsers, and assistive technologies.

#### **Example:**

```
 <!-- Provides alt text for screen readers -->
```

---

#### **b) ARIA (Accessible Rich Internet Applications)**

- Set of **HTML attributes** to enhance accessibility for dynamic content
- Examples: role, aria-label, aria-hidden

#### **Example Code:**

```
<button aria-label="Close Modal">X</button>
```

```
<div role="alert">Form submitted successfully!</div>
```

**Importance:**

- ✓ Makes interactive elements readable by screen readers
  - ✓ Improves accessibility for dynamic content
- 

**c) Section 508 (US Law)**

- US federal standard for accessible electronic content
- Requires **federal websites** to be accessible to people with disabilities

**Importance:**

- ✓ Legal compliance
  - ✓ Protects rights of disabled users
- 

**d) ADA Compliance**

- Americans with Disabilities Act mandates **accessible websites**
  - Includes **perceivable, operable, understandable, robust content**
- 

**3 Practical Accessibility Features**

**a) Keyboard Navigation**

- All interactive elements must be accessible via **keyboard (Tab, Enter, Space)**

**Example Code:**

```
<a href="#main" class="skip-link">Skip to main content</a>
```

---

**b) Color Contrast**

- Text must have **enough contrast** against background

**Example CSS:**

```
body {  
    color: #000; /* black text */  
    background-color: #fff; /* white background */  
}
```

---

**c) Text Alternatives**

- Images, icons, and multimedia must have **alt text or captions**

**Example Code:**

```
  
<video controls>  
  <source src="promo.mp4" type="video/mp4">  
  <track src="promo.vtt" kind="captions" srclang="en" label="English">  
</video>
```

---

**d) Forms Accessibility**

- Label inputs properly
- Provide **error messages**

**Example Code:**

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

---

**e) Semantic HTML**

- Use **proper HTML5 tags**: <header>, <nav>, <main>, <footer>

**Example:**

```
<header>  
  <h1>Website Title</h1>  
  <nav>Navigation Menu</nav>  
</header>  
<main>Page Content</main>  
<footer>© 2025 Company</footer>
```

**Importance:**

- ✓ Helps screen readers understand page structure
  - ✓ Improves SEO
- 

**f) ARIA Roles and Live Regions**

- Indicate **dynamic content updates**

**Example:**

```
<div role="alert">New notification received!</div>
```

---

### g) Responsive Design

- Ensures accessibility across **devices and screen sizes**

#### Example CSS:

```
@media (max-width: 768px) {  
  nav { display: block; }  
}
```

---

### h) Avoiding Accessibility Barriers

- Don't use **color alone to convey information**
  - Avoid **auto-playing audio/video**
  - Provide **skip links**
  - Use **focus states** for interactive elements
- 

#### Testing Accessibility

- Tools: **WAVE, Axe, Lighthouse**
  - Test keyboard navigation, screen readers, and color contrast
- 

#### Key Takeaways

- Web accessibility ensures **inclusive design** for all users
- Follow **WCAG, ARIA, semantic HTML, and responsive design principles**
- Improves **SEO, usability, and legal compliance**
- Integrate accessibility **early in design and development**