**REACT.JS COMPLETE NOTES**

Video 1**:**

**1. What is React?**

* A **JavaScript library** for building **Single Page Applications (SPAs)**.
* Maintained by **Facebook (Meta)**.
* Provides a **smooth user experience** by handling navigation without reloading pages.

**2. Single Page Applications (SPA)**

* Reloads the page only once.
* All further updates & navigation are handled by **JavaScript**.
* Example: Switching pages in a web app without full reload.

**3. Why Learn React?**

* Very popular and in demand.
* Helps in getting jobs in web development.
* Based on **components**:
  + A way to break down a website into reusable chunks.
  + Components can be reused in the same project or across projects.
* Follows **“Write once, use everywhere.”**

**4. Features of React**

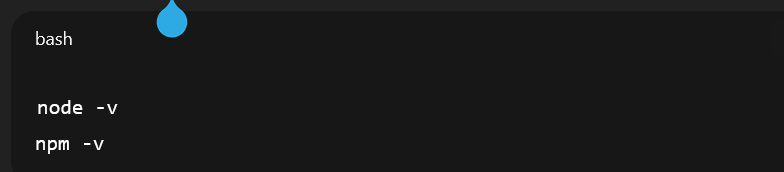
* **Lightweight** – you only install what’s needed.
* **No built-in router** (use react-router-dom or alternatives).
* Works well with other libraries like **Axios** for HTTP requests.
* Easy to start if you know JavaScript basics.

**5. Course Style**

* No prerequisites.
* Includes **JavaScript refresher**.
* Focus on **hands-on learning** with projects.
* Minimal theory → practical coding approach.

**6. Setup & Installation**

**Install Node.js & NPM**

* Node.js = JavaScript runtime to run JS outside browser.
* NPM = Node Package Manager (comes with Node.js).
* Install from [nodejs.org](https://nodejs.org?utm_source=chatgpt.com)
* 

**Install React Dev Tools (Chrome Extension)**

* Inspect React components directly in the browser.
* Shows **props** and **state**.

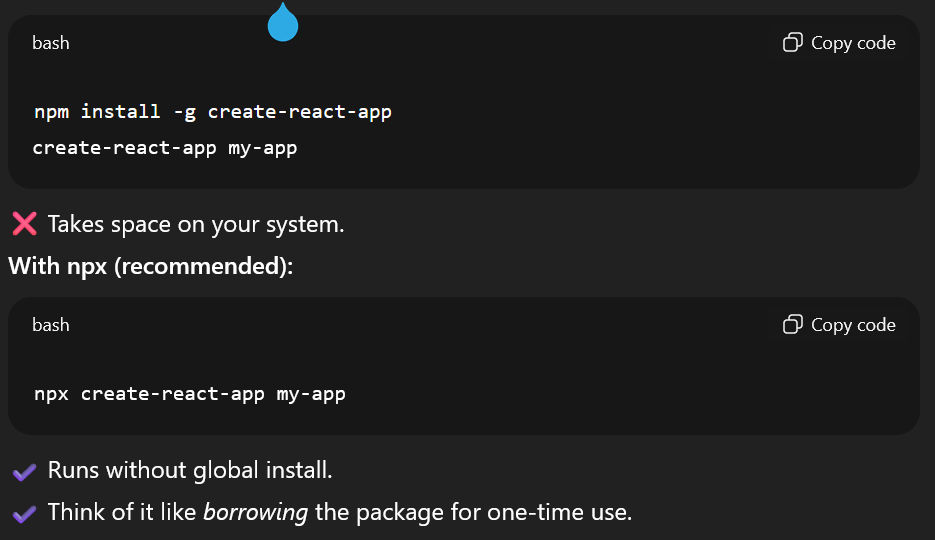
**7. Useful VS Code Extensions**

* **Thunder Client** → Test APIs inside VS Code (like Postman alternative).
* **ES7+ React/Redux/GraphQL/React-Native snippets** → Quick boilerplate code.
* **Bracket Pair Colorizer** → Matches brackets with colors for clarity.
* **Auto Rename Tag** → Auto-updates closing tag when renaming opening tag.
* **Live Server** → Preview static HTML pages in browser with auto-refresh.
* **Prettier (Optional)** → Code formatter.

Video 2**:**

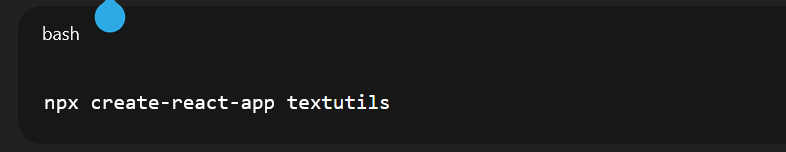
**1. Create React App (CRA)**

* **CRA** = A package that sets up a ready-to-use React project.
* Saves time by generating folder structure + configuration automatically.
* Without CRA → you could set up manually, but it’s not optimal.

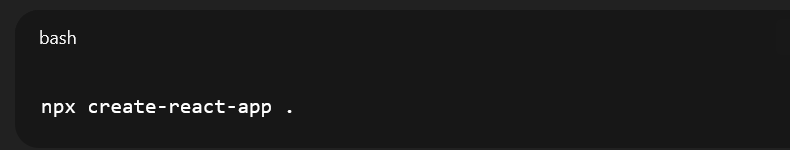
 **Using CRA** With npm (global install):

**Naming Conventions**

* Folder/app name must follow rules (no spaces, must be lowercase, etc.).
* Example:



* To create app in current folder:



**2. React: Library or Framework?**

* Officially → **React is a JavaScript library**.
* But since it competes with Angular/Vue → people often call it a *framework*.

**3. Component-Based Architecture**

* **Component** = Small reusable piece of UI (Navbar, Sidebar, Footer, etc.).
* Benefits:
  + Reusability
  + Easy updates via **props**
  + Smooth navigation & modular design

**4. Props vs State**

* **Props** = External input (like filling details in a form).  
  → Passed from parent → child.  
  → Immutable (cannot be changed by the component itself).
* **State** = Internal data of the component (like heading, content, toggle status).  
  → Component can update its own state.  
  → Changes cause re-render.

⚠️ Props and State may sound similar at first, but you’ll *truly* understand when coding.

**5. CRA Folder Structure**

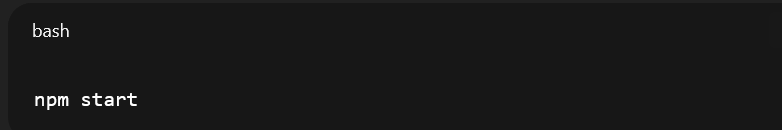
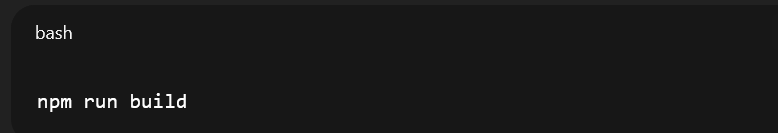
**Important Folders**

* **src/**
  + Contains all your React code (components, logic, styles).
  + Main files:
    - index.js → Entry point, renders <App /> into root.
    - App.js → First main component.
* **public/**
  + Contains index.html (single HTML file).
  + Has a <div id="root"></div> → React injects app here.

**Ignore (for now)**

* node\_modules/ → Contains installed packages (auto-generated).
* package.json & package-lock.json → Store dependencies.
* .gitignore → For GitHub projects.
* README.md → Project description (optional).

**6. Running & Building the App:**

* Start development server: 
* Production build (optimized HTML/CSS/JS):

**7. Class vs Function Components**

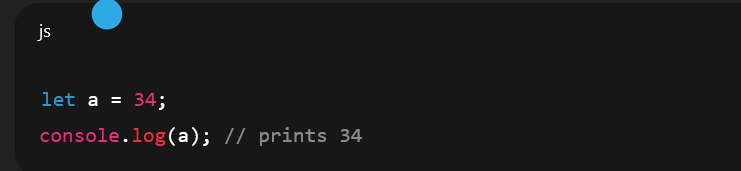
* **Old React**: Mostly class-based.
* **Modern React**: Function-based + Hooks (preferred).

Video 3:

**1. Why JavaScript for React**

* React heavily depends on JavaScript fundamentals.
* Knowing JS basics prevents issues while learning React.

**2. Setup**

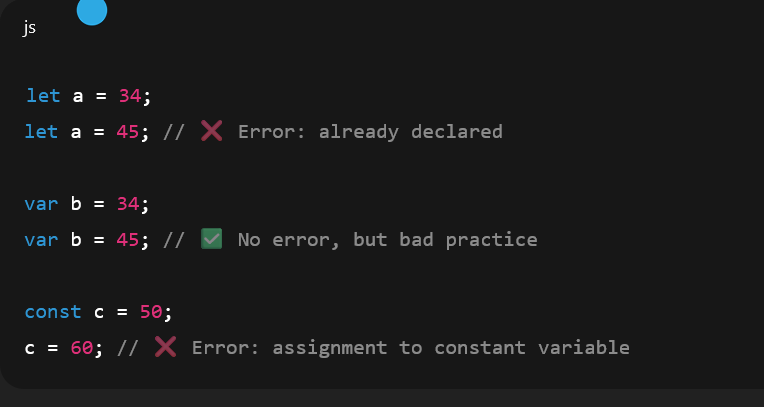
* Create a folder → JS refresher
* Inside, create index.html → add boilerplate (! in VS Code).
* Write JS inside <script> tag.
* Example: 
* Install & use **Live Server** extension in VS Code to run files.

**3. Introduction to JavaScript**

* JS = programming language of the web.
* Originally for client-side, now also used for **server-side** via **Node.js**.
* Ecosystem evolved → Node.js, Express, React, Angular, etc.
* Very popular language today.

**4. Variables in JavaScript**

* **Ways to declare variables**:
  1. var → old, allows redeclaration.
  2. let → introduced in ES6, no redeclaration in same scope.
  3. const → constant, cannot be reassigned.
* Example:



**5. Operators**

* Basic arithmetic: +, -, \*, /
* Example:  
  let x = 10 + 5; // 15

**6. Data Types**

* **Primitive Types**:
  + Number → 10, 3.14
  + String → "Hello"
  + Boolean → true, false
  + Null → null
  + Undefined → undefined
* **Non-primitive**:
  + Object → { key: value }

**7. Functions**

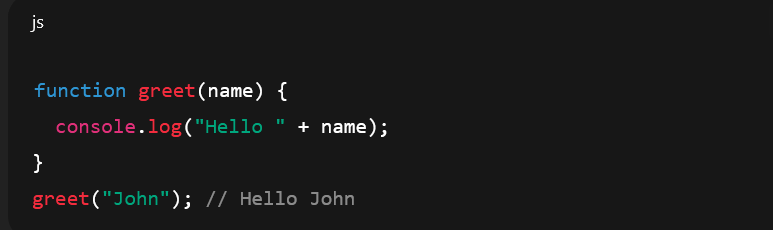
* **Definition & call**:

function hello() {

console.log("Hey Harry");

}

hello(); // Hey Harry

* Functions can take parameters:  
  

8. Objects  


**Arrow function** (ES6):

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

**9. Strings**

* Methods:
  + .slice(start, end)
  + .substring(start, end)
  + .startsWith("text"), .endsWith("text")
  + .indexOf("char")

**10. Arrays**

* Store multiple values:  
  let arr = [1, 2, "hello", true];

**Methods**:

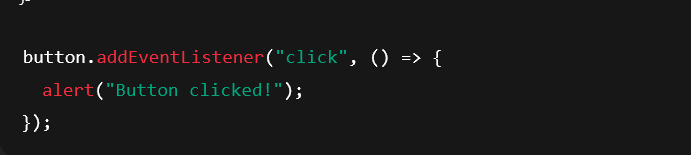
* .forEach() → iterate
* .map() → transform
* .reverse() → reverse order

**11. Loops**

* for, while, do while → same as other languages.
* forEach → common for arrays.
* break / continue → control loop flow.

**12. Events**

* Handle user actions:
  + onclick
  + onmouseover
* Cleaner: addEventListener("event", callback)



**13. Useful Built-ins**

* alert("Hi") → popup.
* confirm("Are you sure?") → OK/Cancel, returns boolean.

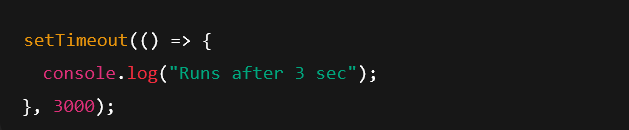
**14. Strict Mode**

* "use strict"; → catches errors (e.g., undeclared variables).

**15. this Keyword**

* In **normal functions**: this = object calling the function.
* In **arrow functions**: this = parent scope (not object).

**16. Callbacks & Asynchronous JS**

* Callback = function passed into another function.
* Example with setTimeout: 
* JS is **non-blocking** → continues executing while waiting.

**Quick Revision Checklist**

* Variables → var, let, const
* Data types → number, string, boolean, undefined, null, object
* Functions → normal vs arrow
* Objects → properties & methods
* Strings & arrays → common methods
* Loops → for, while, forEach
* Events → onclick, addEventListener
* Strict mode → cleaner coding
* this keyword → normal vs arrow
* Callbacks → async execution

Video 4**:**

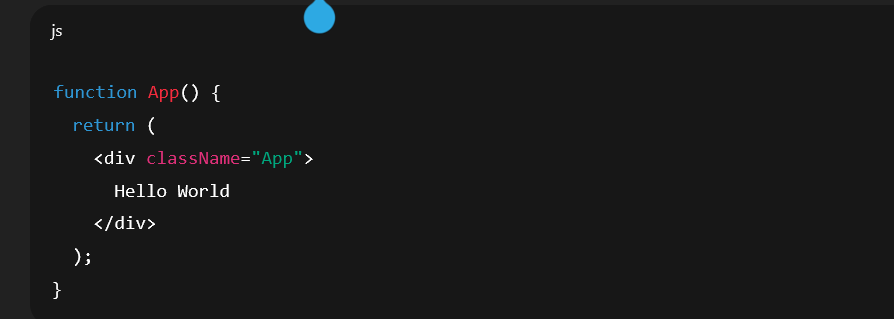
**1. Setup & Basics**

* React apps often created using **Create React App (CRA)**.
* CRA generates folder structure: public/, src/, etc.
* **Entry point**:
  + index.html contains <div id="root"></div>.
  + index.js renders <App /> inside root using ReactDOM.render.
* App.js → your main component; editing this updates the site (localhost:3000).

**2. Why React?**

* **Advantages**:
  + Write less JavaScript, code is more organized.
  + Easy to build **complex apps**.
  + Reusable components (e.g., Navbar).
  + Works seamlessly with APIs.
  + Uses familiar HTML/CSS/JS with slight syntax changes.
* React builds **Single Page Applications (SPAs)**.

**3. Components in React**

* **Two types**:
  + **Function-based components** (modern, simpler, widely used).
  + **Class-based components** (older, less common now).
* Function component example:  
  

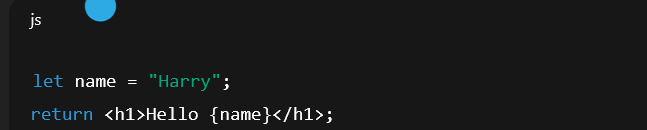
**4. What is JSX?**

* JSX = **JavaScript + XML (HTML-like syntax)**.
* **Definition**: HTML written inside JavaScript with the power to use JS expressions.
* Think: **“HTML wearing the crown of JavaScript.”**
* **Key Features**:
  + Looks like HTML but written in .js files.
  + Can embed JS expressions inside { }.
  + Unified: HTML + JS (and often CSS) in one place.

**5. JSX Rules & Syntax**

* **Must return one parent element**:
  + Wrap multiple elements inside a parent <div> or a **fragment <>...</>**.
* **Attribute differences**:
  + class → className
  + for → htmlFor
  + tabindex → tabIndex (camelCase)
* **Self-closing tags** (e.g., <img />, <input />) must be properly closed.

**6. Using JavaScript in JSX**

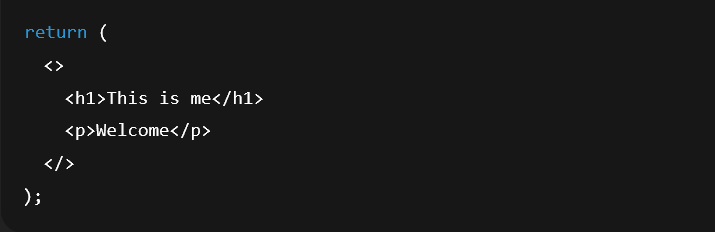
* **Variables**:  
  
* **Expressions**:

Can use operators, conditionals, objects, ternary (condition ? true : false).

**Important**: Do **not** use quotes around { } → it’s directly JavaScript.

* **7. Fragments**

Needed when returning multiple elements without extra wrapper divs.

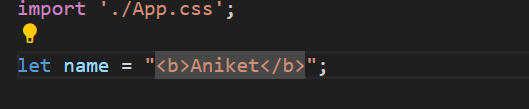


**8. CSS in React**

 Import CSS directly:

* Class selectors applied via className.
* CRA uses **webpack** → automatically bundles CSS.

**9. React’s Security with HTML**

* React **escapes** user input by default → prevents XSS attacks.  
    
  Output:  
  
* Example: passing <b>Aniket</b> as string will render literally, not as HTML.
* Use dangerouslySetInnerHTML only if needed (not recommended).

**10. JSX Compilation (Babel)**

* JSX is **not valid JavaScript** by itself.
* **Babel** compiles JSX → React.createElement() calls.
* Example:  
  

**11. Developer Tips**

* **Use Emmet** in VS Code by enabling "emmet.includeLanguages": { "javascript": "javascriptreact" }.
* Can preview React app on phone via local IP (if on same Wi-Fi).

**12. Key Takeaways**

* JSX is **mostly HTML with slight syntax changes**.
* Always return **a single element**.
* Embed JS inside { }.
* Learn by editing App.js → don’t stress folder structure initially.
* Babel compiles JSX → React understands it.
* Focus now: **write JSX, build small apps, then scale.**