**ReactJS Notes**

### **Section 1: Introduction to React**

React is a JavaScript library developed by Facebook for building user interfaces. It is particularly useful for building single-page applications (SPAs) where UI updates dynamically without full page reloads.

### 🔍 What is the difference between a Library and a Framework?

| Feature | Library | Framework |
| --- | --- | --- |
| Control | You call the library | Framework calls your code |
| Flexibility | High | Low to Medium |
| Examples | React, Lodash | Angular, Django |

React is a **library**: you decide the flow and can integrate it with any tech stack.

### 🔧 Features of React

#### 1. Component-Based Architecture

* Easy to build interfaces
* Reusable components
* Faster development
* Extendable
* Loosely coupled

#### 2. Modular Library

* Can be used only where needed
* Improves app performance
* Lightweight

#### 3. Asynchronous

* Uses Ajax under the hood
* Updates only required sections of the page

#### 4. Virtual DOM

* A virtual representation of the UI in memory

### 🌳 What is DOM?

DOM stands for **Document Object Model** — it’s a tree structure of HTML elements loaded in the browser.

#### 🧠 Browser Engine Examples

| Browser | Rendering Engine | JavaScript Engine |
| --- | --- | --- |
| Chrome | Blink | V8 |
| Firefox | Gecko | SpiderMonkey |
| Safari | WebKit | Nitro |
| Edge | Chromium | Chakra |

### 🏗 Architecture of a Browser

1. UI Layer
2. Browser Engine
3. Rendering Engine
4. Networking
5. JavaScript Interpreter
6. UI Backend
7. Data Storage

#### 🌐 Browser Workflow

Markup **→** Bytes → Characters → Tokens → Nodes → DOM → Layout → Render → Paint → Display

* Separate for Styles:

CSS → Bytes → Characters → CSSOM → Layout → Render → Paint

### 🌳 Shadow DOM vs Virtual DOM vs DOM

| Type | Description | Diagram |
| --- | --- | --- |
| DOM | Tree of all HTML elements in the document | <html><body><div><p>Hello</p></div></body></html> |
| Shadow DOM | Localized DOM tree inside an element | <custom-element> <#shadow-root> <p>Hello</p> </custom-element> |
| Virtual DOM | JS representation of DOM for efficient updates | { tag: 'div', children: [{ tag: 'p', text: 'Hello' }] } |

#### 🧪 Differences

* **DOM**: Full document
* **Shadow DOM**: Scoped to component
* **Virtual DOM**: Used for fast re-rendering

### ❌ Cons of React

* Only handles UI, needs help from other tools/libraries
* Doesn’t manage full app structure
* Form & validation handled by external libs

### 🔌 Common 3rd-Party React Libraries

#### 🧾 Forms & Validation

* Formik, React Hook Form, Final Form
* Yup, Zod, Validator.js

#### 🌐 API Handling

* Axios, Fetch, React Query, SWR, SuperAgent

#### 🗺 Routing

* React Router DOM, Wouter

#### 🎨 UI Libraries

* Material UI, React Bootstrap, Ant Design, Chakra UI

#### 🔧 State Management

* Redux, Zustand, Recoil, MobX

#### 🎞 Animations

* Framer Motion, React Spring, GSAP

#### 📅 Utilities & Time

* Lodash, Day.js

#### ✅ Testing

* Jest, React Testing Library, Enzyme

#### 📈 Charts

* Chart.js, Recharts, Victory

#### 📂 File Uploads & Icons

* React Dropzone, Uploady, React Icons, Heroicons

### 🧠 Mini Quiz: React Basics & Browser Architecture

1. **True or False**: React is a full-fledged framework like Angular.
2. **Fill in the Blank**: React uses a \_\_\_\_ DOM to improve UI update speed.
3. **Multiple Choice**: Which of these is **NOT** a browser rendering engine? a) Blink b) Nitro c) Gecko d) Chakra UI
4. **Short Answer**: What is the main advantage of using a Virtual DOM over a traditional DOM?
5. **Real World**: You’re building a form-heavy app with third-party validation. Which two React libraries might you use for form handling and validation?

### **Section 2: React Environment Setup & Toolchain**

🧭 Why Environment Setup Matters

Modern React apps need:

* A JavaScript runtime to run build tools (Node.js)
* A package manager to install libraries (npm or yarn)
* A compiler to convert JSX into browser-ready JavaScript (Babel)
* A module bundler to bundle JavaScript, styles, assets (Webpack or Vite)
* A local dev server to preview apps (webpack-dev-server or Vite built-in)

### ⚙️ Core Tools & What They Do

| Tool | Purpose |
| --- | --- |
| Node.js | Runs build tools outside the browser |
| npm / yarn | Installs libraries and manages dependencies |
| Babel | Transforms JSX and ES6+ into compatible JS |
| Webpack | Bundles JS, CSS, images into optimized files |
| VS Code | Code editor for development |

### 🛠 Option 1: Using Create React App (CRA)

npx create-react-app my-app  
cd my-app  
npm start

Folder Structure:

my-app/  
├── node\_modules/  
├── public/  
│ └── index.html  
├── src/  
│ ├── App.js  
│ └── index.js  
├── package.json  
└── README.md

### ⚡ Option 2: Using Vite (Fast & Modern)

npm create vite@latest my-app -- --template react  
cd my-app  
npm install  
npm run dev

| Feature | CRA | Vite |
| --- | --- | --- |
| Startup Speed | Slower | ⚡ Super Fast |
| Dev Server | Webpack | Native with ESBuild |
| Bundler | Webpack | Rollup |
| Setup Complexity | Low | Low |
| TypeScript | Optional | Easy via template |

### 🛠 Manual Setup (For Legacy or Partial React Apps)

Add React via CDN:

<script src="https://unpkg.com/react@17/umd/react.development.js"></script>  
<script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>  
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>

Use JSX inline:

<script type="text/babel">  
 ReactDOM.render(<h1>Hello!</h1>, document.getElementById('root'));  
</script>

### 🧰 Toolchain Summary

| Category | Recommended Tools |
| --- | --- |
| Compiler | Babel |
| Bundler | Webpack / Vite |
| Dev Server | webpack-dev-server / Vite |
| Package Manager | npm / yarn |
| Editor | VS Code |

### 🧠 Mini Quiz: Section 2 — Environment & Toolchain

1. **True or False**: Vite uses Babel to compile JSX.
2. **Fill in the Blank**: create-react-app uses \_\_\_\_ as its module bundler.
3. **Multiple Choice**: What is the fastest dev setup for React in 2025? a) CRA b) Webpack c) Vite d) Babel
4. **Short Answer**: What’s the difference between CRA and Vite in terms of build tools?
5. **Real World**: You’re integrating React into a legacy Spring Boot web app. Should you use CRA, Vite, or a manual setup? Why?

### **Section 3: JSX & Rendering Elements**

**🔷 What is JSX?**

JSX stands for **JavaScript XML**. It lets you write HTML-like syntax directly inside JavaScript, which React then transpiles to JavaScript using Babel.

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

This code is transformed into:

const element = React.createElement('h1', null, 'Hello, JSX!');

✅ JSX is syntactic sugar — it makes code more readable and expressive.

### 🧱 Rendering Elements in React

To render an element into the DOM, use:

const element = <h1>Hello, world!</h1>;  
ReactDOM.render(element, document.getElementById('root'));

### 🚨 JSX Rules to Remember

| Rule | Example |
| --- | --- |
| Must return a single parent tag | ✅ <div><h1></h1><p></p></div> |
| Use className instead of class | ✅ <div className="box"> |
| Use camelCase for attributes | ✅ onClick, readOnly, tabIndex |
| Embed expressions in {} | ✅ <h1>{user.name}</h1> |
| Self-closing tags need / | ✅ <img />, <input />, <br /> |

### 💡 Why Use JSX?

* Improves code readability
* Prevents XSS attacks via automatic escaping
* Closer integration of logic and layout

### 🧬 JSX Internals Diagram (Conceptual)

JSX  
 ↓  
Babel  
 ↓  
React.createElement()  
 ↓  
JavaScript Object  
 ↓  
Virtual DOM  
 ↓  
Real DOM

### 🧮 Example: Embedding Variables & Functions

const user = { name: "Sai", age: 25 };  
function formatUser(u) {  
 return `${u.name} (${u.age})`;  
}  
const element = <h2>Hello, {formatUser(user)}</h2>;

### 🔁 JSX with Conditional Rendering

const isLoggedIn = true;  
const element = (  
 <div>  
 {isLoggedIn ? <p>Welcome back!</p> : <p>Please sign in.</p>}  
 </div>  
);

### 🗂 JSX with Lists

const items = ['React', 'Angular', 'Vue'];  
const list = (  
 <ul>  
 {items.map((tech, index) => <li key={index}>{tech}</li>)}  
 </ul>  
);

### ⚠️ Common Mistakes in JSX

| Mistake | Correct Usage |
| --- | --- |
| Using class | Use className |
| Not using unique key in list | Use key={index} in .map() |
| Using if instead of ternary | Use { condition ? ... : ... } |
| Not wrapping multiple elements | Use <div> or <>...</> |

### 🧠 Mini Quiz: JSX & Rendering

1. **True or False**: JSX is a string template that’s rendered directly as HTML.
2. **Fill in the Blank**: You must wrap JSX expressions in \_\_\_\_\_\_\_\_ brackets.
3. **Multiple Choice**: Which of the following is not valid JSX syntax?  
   1. <h1>Hello</h1>
   2. <div class='box'>
   3. <img src='img.jpg' />
   4. <input type='text' />
4. **Short Answer**: Why is using key important in lists?

### 🌍 Real World Scenario

You’re building a dashboard showing dynamic user data. JSX helps bind dynamic names, scores, and conditional colors right in the markup — making rendering intuitive and less error-prone.

### 💬 Interview Tip

“JSX is not HTML. Explain how Babel compiles JSX to React.createElement() and how React converts that to Virtual DOM. Also, clarify how keys help optimize re-renders.”

### **Section 4: Components in React**

React is all about components — reusable, independent pieces of UI that can be composed to build complex interfaces.

### 🧩 What is a Component?

A **component** is a JavaScript function or class that optionally accepts inputs (called props) and returns a React element that describes how a section of the UI should appear.

There are two main types:

* **Function Components** (preferred)
* **Class Components**

### 📘 Function Component Example

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

Usage:

<Welcome name="Sai" />

### 🧾 Class Component Example

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

### 🏗 Component Tree Example

<App>  
 <Header />  
 <Sidebar />  
 <Main>  
 <Article />  
 <Footer />  
 </Main>  
</App>

React components form a **tree structure**, making it easy to manage layouts.

### 🪄 Props in React

* Props (short for **properties**) are read-only inputs to a component.
* Passed from parent to child.

function User(props) {  
 return <p>{props.name} - {props.age} years old</p>;  
}

### 🧠 Key Differences: Function vs Class Components

| Feature | Function Component | Class Component |
| --- | --- | --- |
| Syntax | Simpler | Verbose |
| Lifecycle | Use useEffect() | Has lifecycle methods like componentDidMount() |
| State Support | Via useState() | this.state, this.setState() |
| Performance | Slightly better | Slightly heavier |

### ♻️ Component Reusability

Components should:

* Be **small and focused**
* Accept **props**
* Avoid directly modifying props (immutable)
* Return **valid JSX**

### 🧠 Mini Quiz: Components in React

1. **True or False**: Props can be modified by the child component.
2. **Fill in the Blank**: A class component uses this.\_\_\_ to access props.
3. **Multiple Choice**: Which hook is used to manage state in a function component?
   * 1. useEffect
     2. useProps
     3. useState
     4. componentDidMount
4. **Short Answer**: What’s one benefit of using function components over class components?
5. **Real World**: You want to build a reusable Card component that accepts title, description, and image as props. How would you structure it?

### **Section 4: Components & Props**

Components are the core building blocks of any React app. They split the UI into reusable, isolated pieces of code that represent part of the user interface.

#### 🧱 What is a Component?

A component is a JavaScript function or class that optionally accepts inputs (called “props”) and returns a React element that describes how a section of the UI should appear.

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

#### Types of Components

| Type | Description |
| --- | --- |
| Functional | Uses plain functions and hooks |
| Class-based | Uses ES6 classes and lifecycle methods |

// Functional Component  
function Greet() {  
 return <h2>Hello from Function</h2>;  
}  
  
// Class Component  
class Greet extends React.Component {  
 render() {  
 return <h2>Hello from Class</h2>;  
 }  
}

#### Props (Short for Properties)

Props are read-only attributes passed to a component to customize its behavior or display.

function Greeting(props) {  
 return <p>Welcome, {props.name}</p>;  
}  
  
<Greeting name="Sai" />

* Props are immutable.
* Props allow components to be reused with different data.

#### Composing Components

Components can contain other components to create a hierarchy:

function App() {  
 return (  
 <div>  
 <Greeting name="Sai" />  
 <Greeting name="Peta" />  
 </div>  
 );  
}

### 🔄 Component Reusability and Best Practices

* Use **props** to customize output.
* Use **defaultProps** to set default values.
* Keep components **pure** — don’t modify props inside.

Greeting.defaultProps = {  
 name: 'Guest'  
};

### 🧠 Mini Quiz: Components & Props

1. **True or False**: Props are mutable and can be changed inside a component.
2. **Fill in the Blank**: A component must return a \_\_\_\_\_\_.
3. **Multiple Choice**: What is the key difference between a class and a functional component? a) Class components are faster  
   1. Functional components use hooks
   2. Class components cannot use props
   3. Functional components have render()
4. **Short Answer**: How do you pass data from a parent to a child component in React?
5. **Real World**: You want to create a profile card that displays a user’s name, age, and picture. Would you use props or state to pass the user’s data into the card component?

### **Section 5: State & Lifecycle**

In React, **state** refers to an object that determines how a component behaves or appears. State allows components to create and manage their own data internally.

#### 🧠 What is State?

State is managed within the component (unlike props which are passed from parent).

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>  
 );  
}

#### Class Component Equivalent:

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

### 🔁 Lifecycle Methods (Class Components Only)

| Method | When It Runs |
| --- | --- |
| constructor | When component is initialized |
| render | When component is rendered |
| componentDidMount | After first render |
| componentDidUpdate | After every update (except first render) |
| componentWillUnmount | Just before the component is removed |

class MyComponent extends React.Component {  
 componentDidMount() {  
 console.log('Component mounted');  
 }  
  
 componentWillUnmount() {  
 console.log('Component will unmount');  
 }  
  
 render() {  
 return <div>Hello!</div>;  
 }  
}

### ⚙️ Lifecycle in Functional Components (Using Hooks)

import { useEffect } from 'react';  
  
function Demo() {  
 useEffect(() => {  
 console.log("Mounted");  
 return () => console.log("Unmounted");  
 }, []);  
  
 return <p>Lifecycle Example</p>;  
}

### 🧠 Mini Quiz: State & Lifecycle

1. **True or False**: Props can be changed using useState().
2. **Fill in the Blank**: The function used to update state in a class component is called \_\_\_\_\_\_\_\_\_.
3. **Multiple Choice**: What lifecycle method is used to fetch data on first render? a) componentDidUpdate  
   1. constructor
   2. componentDidMount
   3. render
4. **Short Answer**: What is the difference between props and state in React?
5. **Real World**: You’re building a to-do app. Where would you store the list of tasks — props or state? Why?

### **Section 6: Handling Events**

In React, event handling closely resembles handling events in the DOM, but with a few important differences:

### 🖱️ Key Concepts:

* Event names use **camelCase** (onClick, onChange, etc.) instead of lowercase.
* You pass **functions** as event handlers rather than strings.
* Events are **synthetic** — React wraps the browser’s native event system for better performance and cross-browser compatibility.

### 🛠 Example: Functional Component

function Greet() {  
 function handleClick() {  
 alert("Hello!");  
 }  
 return <button onClick={handleClick}>Click Me</button>;  
}

### 🔧 Example: Binding in Class Component

class Clicker extends React.Component {  
 constructor() {  
 super();  
 this.state = { count: 0 };  
 this.handleClick = this.handleClick.bind(this);  
 }  
  
 handleClick() {  
 this.setState({ count: this.state.count + 1 });  
 }  
  
 render() {  
 return <button onClick={this.handleClick}>Click Me</button>;  
 }  
}

### ✅ Common Event Types

* onClick
* onChange
* onSubmit
* onMouseEnter, onMouseLeave
* onKeyDown, onKeyUp
* onFocus, onBlur

### 🔄 Event Handling with Parameters

function Welcome(props) {  
 function greet(name) {  
 alert("Welcome, " + name);  
 }  
  
 return <button onClick={() => greet("Sai")}>Say Hi</button>;  
}

### 🔥 Prevent Default Behavior

function Form() {  
 function handleSubmit(e) {  
 e.preventDefault();  
 console.log("Form Submitted");  
 }  
  
 return <form onSubmit={handleSubmit}><button type="submit">Submit</button></form>;  
}

### 🧠 Mini Quiz: Section 6 — Handling Events

1. **True or False**: React event handlers accept strings like in HTML (onClick="doSomething()").
2. **Fill in the Blank**: Event names in JSX are written in \_\_\_\_\_\_\_\_ case.
3. **Multiple Choice**: Which is the correct way to bind this in a class component? a) this.handleClick.bind() b) handleClick.bind(this) c) this.handleClick = this.handleClick.bind(this) d) No need to bind at all
4. **Short Answer**: What is the purpose of e.preventDefault() in event handlers?
5. **Real World**: You want to call a function with a parameter when a button is clicked. How would you do it?

### **Section 7: Conditional Rendering**

In React, conditional rendering means displaying different UI based on application state or logic.

### 🔁 Key Techniques

#### ✅ Using if/else Outside JSX

function Greeting(props) {  
 const isLoggedIn = props.isLoggedIn;  
 if (isLoggedIn) {  
 return <h1>Welcome back!</h1>;  
 }  
 return <h1>Please sign in.</h1>;  
}

#### ❓ Ternary Operator Inside JSX

<h1>{isLoggedIn ? 'Welcome back!' : 'Please sign in.'}</h1>

#### ✔️ Logical AND (&&) Operator

{messages.length > 0 && <h2>You have {messages.length} new messages.</h2>}

#### 🛠 Conditional Component Rendering

function UserGreeting() {  
 return <h1>Welcome Back!</h1>;  
}  
function GuestGreeting() {  
 return <h1>Please Sign Up</h1>;  
}  
function Greeting(props) {  
 const isLoggedIn = props.isLoggedIn;  
 return isLoggedIn ? <UserGreeting /> : <GuestGreeting />;  
}

### ⚡ Advanced Example: Conditional Button

function LoginButton(props) {  
 return <button onClick={props.onClick}>Login</button>;  
}  
function LogoutButton(props) {  
 return <button onClick={props.onClick}>Logout</button>;  
}

Use it conditionally:

function LoginControl() {  
 const [isLoggedIn, setIsLoggedIn] = useState(false);  
  
 const handleLogin = () => setIsLoggedIn(true);  
 const handleLogout = () => setIsLoggedIn(false);  
  
 const button = isLoggedIn ? (  
 <LogoutButton onClick={handleLogout} />  
 ) : (  
 <LoginButton onClick={handleLogin} />  
 );  
  
 return (  
 <div>  
 {button}  
 <Greeting isLoggedIn={isLoggedIn} />  
 </div>  
 );  
}

### 🧠 Mini Quiz: Section 7 — Conditional Rendering

1. **True or False**: You can use if directly inside JSX blocks.
2. **Fill in the Blank**: The ternary syntax for conditions is condition ? A : B. This is used \_\_\_\_ JSX.
3. **Multiple Choice**: Which of the following are valid ways to render conditional content in React?  
   1. if/else inside JSX
   2. Ternary inside JSX
   3. && logic
   4. All of the above
4. **Short Answer**: Why can’t we use traditional if statements directly inside JSX?
5. **Real World**: How would you conditionally show a login form only if the user is **not** authenticated?

### **Section 8: Forms in React**

In React, forms are handled using **controlled components**, where form data is handled by the React component’s state.

#### 🧾 Basic Controlled Component

import React, { useState } from 'react';  
  
function MyForm() {  
 const [name, setName] = useState('');  
  
 function handleSubmit(e) {  
 e.preventDefault();  
 alert(`Hello, ${name}`);  
 }  
  
 return (  
 <form onSubmit={handleSubmit}>  
 <label>Name: </label>  
 <input value={name} onChange={(e) => setName(e.target.value)} />  
 <button type="submit">Submit</button>  
 </form>  
 );  
}

### 📥 Input Types

* Text, Email, Password, Radio, Checkbox, Select, Textarea

Example:

<input type="email" value={email} onChange={handleChange} />  
<textarea value={message} onChange={handleChange} />

### 🧰 Handling Multiple Inputs

function Form() {  
 const [formData, setFormData] = useState({ username: '', email: '' });  
  
 function handleChange(e) {  
 setFormData({ ...formData, [e.target.name]: e.target.value });  
 }  
  
 return (  
 <>  
 <input name="username" value={formData.username} onChange={handleChange} />  
 <input name="email" value={formData.email} onChange={handleChange} />  
 </>  
 );  
}

### ✅ Form Validation (Using Third-Party)

**Formik + Yup** is a powerful combination:

npm install formik yup

import { useFormik } from 'formik';  
import \* as Yup from 'yup';  
  
const formik = useFormik({  
 initialValues: { email: '' },  
 validationSchema: Yup.object({  
 email: Yup.string().email().required()  
 }),  
 onSubmit: values => console.log(values)  
});

### 🧠 Mini Quiz: Section 8 — Forms

1. **Fill in the Blank**: A form where input value is tracked by React is called a \_\_\_\_ component.
2. **True/False**: You should let the DOM manage input values instead of React.
3. **Multiple Choice**: Which library is commonly used for form validation in React? a) Redux  
   1. Formik
   2. Vite
   3. Bootstrap
4. **Short Answer**: What is the purpose of e.preventDefault() in a form submit handler?
5. **Real World**: You’re building a multi-input registration form. What technique helps avoid writing separate useState for each input?

### **Section 9: React Hooks**

React Hooks let you use state and other React features without writing a class.

#### 🔧 What are Hooks?

Hooks are built-in functions introduced in React 16.8 that allow you to “hook into” React state and lifecycle features from function components.

### 🔹 Commonly Used Hooks

| Hook | Purpose |
| --- | --- |
| useState | Add state to functional components |
| useEffect | Run side-effects (fetching, subscriptions, etc.) |
| useRef | Access and persist mutable DOM or values |
| useContext | Access context API values |

### 🧪 Example: useState

import React, { useState } from 'react';  
  
function Counter() {  
 const [count, setCount] = useState(0);  
  
 return (  
 <div>  
 <p>You clicked {count} times</p>  
 <button onClick={() => setCount(count + 1)}>Click me</button>  
 </div>  
 );  
}

* useState(0) initializes the state variable count to 0.
* setCount is used to update the value.

### 🧪 Example: useEffect

import React, { useState, useEffect } from 'react';  
  
function Timer() {  
 const [seconds, setSeconds] = useState(0);  
  
 useEffect(() => {  
 const interval = setInterval(() => {  
 setSeconds((prev) => prev + 1);  
 }, 1000);  
  
 return () => clearInterval(interval); // cleanup  
 }, []);  
  
 return <h2>Time: {seconds}s</h2>;  
}

* The empty array [] tells React to run useEffect only once (componentDidMount).

### 🧪 Example: useRef

import React, { useRef } from 'react';  
  
function InputFocus() {  
 const inputRef = useRef();  
  
 function handleClick() {  
 inputRef.current.focus();  
 }  
  
 return (  
 <>  
 <input ref={inputRef} type="text" />  
 <button onClick={handleClick}>Focus Input</button>  
 </>  
 );  
}

* useRef() gives a reference to the DOM node.

### 🔍 useEffect Lifecycle Diagram

Mount: useEffect() → DOM update → runs once  
Update: dependencies change → effect re-run  
Unmount: return cleanup function

### 🧠 Mini Quiz: Section 9 — Hooks

1. **True or False**: useEffect runs before the first render.
2. **Fill in the Blank**: useState returns a value and a \_\_\_\_ function.
3. **Multiple Choice**: What does an empty dependency array [] mean? a) Run on every render  
   1. Never run
   2. Run once on mount
   3. Run on unmount
4. **Short Answer**: When should you use useRef?
5. **Real World**: You want to run code when the component mounts, like fetching user data. What hook would you use?

### **Section 10: React Router (Basics)**

React Router is a standard routing library for React that enables navigation among views of various components in a React Application, using browser’s history.

#### 📦 Installation

To install React Router DOM:

npm install react-router-dom

#### 🚏 Key Components

| Component | Description |
| --- | --- |
| <BrowserRouter> | Enables dynamic routing using the HTML5 history API |
| <Routes> | Container for all routes |
| <Route> | Defines a path-to-component mapping |
| <Link> | Used for client-side navigation |
| <NavLink> | Like <Link>, but with active styling support |
| useNavigate() | Hook for programmatic navigation |

#### 🔧 Example Usage

import { BrowserRouter, Routes, Route, Link } from 'react-router-dom';  
import Home from './Home';  
import About from './About';  
  
function App() {  
 return (  
 <BrowserRouter>  
 <nav>  
 <Link to="/">Home</Link>  
 <Link to="/about">About</Link>  
 </nav>  
 <Routes>  
 <Route path="/" element={<Home />} />  
 <Route path="/about" element={<About />} />  
 </Routes>  
 </BrowserRouter>  
 );  
}

#### 🧠 Mini Quiz: React Router Basics

1. **True or False**: React Router DOM allows full page reloads on navigation.
2. **Fill in the Blank**: To define a path-component pair, use the <\_\_\_\_> tag inside <Routes>.
3. **Multiple Choice**: Which component provides navigation with active link highlighting?
4. <Link>
5. <NavLink>
6. <BrowserRouter>
7. <Route>
8. **Short Answer**: What does the useNavigate() hook allow you to do?
9. **Real World**: You want to redirect users to a login page after logout. How would you handle that using React Router?

### **Section 11: Advanced Hooks & Performance Optimization**

React offers additional hooks and techniques beyond the basic useState and useEffect, aimed at more complex logic and performance tuning.

#### 🚀 Common Advanced Hooks

| Hook | Purpose |
| --- | --- |
| useReducer | For complex state logic, like Redux-style reducers |
| useCallback | Memoize functions to avoid unnecessary re-renders |
| useMemo | Memoize expensive calculations |
| useRef | Create mutable refs that persist across renders |
| useLayoutEffect | Like useEffect but runs synchronously after DOM updates |
| useImperativeHandle | Customize the instance value exposed by ref |

#### 🔄 useReducer Example

const initialState = { count: 0 };  
  
function reducer(state, action) {  
 switch (action.type) {  
 case 'increment': return { count: state.count + 1 };  
 case 'decrement': return { count: state.count - 1 };  
 default: return state;  
 }  
}  
  
function Counter() {  
 const [state, dispatch] = useReducer(reducer, initialState);  
 return (  
 <>  
 <p>{state.count}</p>  
 <button onClick={() => dispatch({ type: 'increment' })}>+</button>  
 <button onClick={() => dispatch({ type: 'decrement' })}>-</button>  
 </>  
 );  
}

#### ⚙️ Performance Tips

* Use React.memo() for functional components
* Split code using dynamic imports (React.lazy, Suspense)
* Debounce heavy operations (e.g., with Lodash)
* Optimize lists with key and useMemo where needed

#### 🧠 Mini Quiz: Advanced Hooks & Optimization

1. **True or False**: useCallback prevents function recreation on re-renders.
2. **Fill in the Blank**: useReducer returns a state and a \_\_\_\_\_\_ function.
3. **Multiple Choice**: Which hook helps delay code execution after layout? a) useEffect  
   1. useMemo
   2. useLayoutEffect
   3. useRef
4. **Short Answer**: When should you use React.memo() in a component?
5. **Real World**: Your form has performance issues due to many re-renders. Which hooks or techniques can help reduce re-renders?

### **Section 12: Mini Project (Bootstrap + React)**

This section walks through a basic mini project that combines React with Bootstrap for styling.

#### 📌 Objective

Build a simple **To-Do List Application** using React with Bootstrap components.

#### 🛠 Technologies Used

* React (with Create React App or Vite)
* Bootstrap 5 (via CDN or npm)

#### 🧱 App Structure

TodoApp/  
├── App.js  
├── TodoList.js  
├── TodoItem.js  
└── AddTodo.js

#### 📦 Install Bootstrap (Optional)

npm install bootstrap

Then import it in index.js:

import 'bootstrap/dist/css/bootstrap.min.css';

#### 💡 App.js

import React, { useState } from 'react';  
import AddTodo from './AddTodo';  
import TodoList from './TodoList';  
  
function App() {  
 const [todos, setTodos] = useState([]);  
  
 const addTodo = (task) => {  
 const newTodo = { id: Date.now(), task, completed: false };  
 setTodos([...todos, newTodo]);  
 };  
  
 return (  
 <div className="container mt-5">  
 <h1 className="text-center">React To-Do App</h1>  
 <AddTodo onAdd={addTodo} />  
 <TodoList todos={todos} />  
 </div>  
 );  
}  
  
export default App;

#### 💡 AddTodo.js

import React, { useState } from 'react';  
  
function AddTodo({ onAdd }) {  
 const [task, setTask] = useState('');  
  
 const handleSubmit = (e) => {  
 e.preventDefault();  
 if (!task.trim()) return;  
 onAdd(task);  
 setTask('');  
 };  
  
 return (  
 <form onSubmit={handleSubmit} className="d-flex mb-3">  
 <input className="form-control me-2" value={task} onChange={(e) => setTask(e.target.value)} />  
 <button className="btn btn-primary">Add</button>  
 </form>  
 );  
}  
  
export default AddTodo;

#### 💡 TodoList.js

import React from 'react';  
import TodoItem from './TodoItem';  
  
function TodoList({ todos }) {  
 return (  
 <ul className="list-group">  
 {todos.map((todo) => (  
 <TodoItem key={todo.id} todo={todo} />  
 ))}  
 </ul>  
 );  
}  
  
export default TodoList;

#### 💡 TodoItem.js

import React from 'react';  
  
function TodoItem({ todo }) {  
 return <li className="list-group-item">{todo.task}</li>;  
}  
  
export default TodoItem;

#### 🧠 Mini Quiz: React + Bootstrap Project

1. **True or False**: You can use Bootstrap with React through both CDN and npm.
2. **Fill in the Blank**: The useState hook returns a value and a \_\_\_\_ function.
3. **Multiple Choice**: What is used to group each todo item visually in Bootstrap?  
   1. .container
   2. .form-control
   3. .list-group-item
   4. .card
4. **Short Answer**: How would you handle editing or deleting a to-do item in this app?
5. **Real World**: Suggest one enhancement to improve UX in this mini project.