React Notes (Plain Text)

--- Overview ---
React is a declarative, component based JavaScript library for building user interfaces. It uses a virtual DOM to efficiently update the real DOM.

--- Core Concepts ---
Component: Reusable UI unit. Can be a function (Function Component) or a class (Class Component).
JSX: Syntax extension that looks like HTML; transpiled to React.createElement calls.
Props: Read only inputs passed from parent to child.
State: Mutable data local to a component; triggers re render on change.

--- Function Components ---
- Defined as a JavaScript function that returns JSX.
- Use React Hooks for state, side effects, refs, etc.

--- Class Components ---
- Extend React.Component.
- Have a constructor, render method, and lifecycle methods (componentDidMount, etc.).
- Less common after Hooks were introduced.

--- Hooks ---
useState(initial): Returns [value, setValue].
useEffect(effect, deps?): Runs side effects; deps array controls when it runs.
useContext(Context): Accesses nearest Context value.

useReducer(reducer, init): State management with reducer pattern.
useRef(initial): Mutable ref object; persists across renders.
useMemo(fn, deps): Memoizes expensive calculation.
useCallback(fn, deps): Memoizes function identity.

--- Lifecycle (Function Components via useEffect) ---
Mount: useEffect(() => { ... }, [])
Update: useEffect(() => { ... }, [deps])
Unmount: return () => { cleanup } from useEffect.

--- Context ---
1. Create: const MyContext = React.createContext(defaultValue);
2. Provider: <MyContext.Provider value={...}>children</MyContext.Provider>
3. Consumer: useContext(MyContext) or <MyContext.Consumer>.

--- Routing ---
- React Router library (v6+).
- BrowserRouter wraps app.
- Routes: <Routes><Route path="/home" element={<Home/>} /></Routes>.

--- State Management ---
- Local: useState / useReducer.
- Global: Context API, Redux, Zustand, Recoil, Jotai, etc.

--- Performance Optimizations ---

React.memo(Component): Prevent re render if props unchanged.
useMemo / useCallback: Memoize values and functions.
Lazy loading: const LazyComp = React.lazy(() => import('./Comp'));
Suspense: <Suspense fallback={...}> <LazyComp/> </Suspense>.

--- Event Handling ---
- Synthetic events (cross browser wrapper).
- camelCase prop names: onClick, onChange.
- Prevent default: e.preventDefault().

--- Forms ---
- Controlled: value bound to state, onChange updates state.
- Uncontrolled: useRef to access DOM node values.

--- Styling ---
- CSS Modules, Styled Components, Emotion, Tailwind, inline styles, CSS in JS.

--- Testing ---
- Jest + React Testing Library for unit/component tests.
- Enzyme (legacy) for shallow rendering.
- Cypress for end to end testing.

--- Common Patterns ---
Higher Order Component (HOC): function that returns a new component.
Render Props: prop that is a function returning JSX.

Compound Components: parent controls shared state for child components.

--- Project Structure (Typical) ---
src/
  components/   // reusable UI pieces
  pages/   // route level components
  hooks/   // custom hooks
  context/   // context providers
  utils/   // helper functions
  services/   // API calls
  App.jsx
  index.jsx

--- Build Tools ---
- Create React App (CRA) for zero config setup.
- Vite, Next.js, Remix for advanced bundling and SSR.

--- Important Packages ---
react, react-dom, react-router-dom, redux/@reduxjs/toolkit, @testing-library/react, styled-components, axios/fetch.

--- Tips ---
1. Keep components small and focused on a single responsibility.
2. Prefer function components + hooks over class components.
3. Use PropTypes or TypeScript for type safety.
4. Memoize expensive calculations, not premature.
5. Keep side effects inside useEffect; avoid direct DOM manipulation.

6. Follow the "single source of truth" principle for state.

--- Quick Reference Snippets ---

```
// Function component with state & effect
function Counter() {
  const [count, setCount] = React.useState(0);
  React.useEffect(() => {
    console.log('Mounted or count changed');
    return () => console.log('Cleanup');
  }, [count]);
  return (
    <button onClick={() => setCount(c => c + 1)}>
      Count: {count}
    </button>
  );
}

// Custom hook example
function useFetch(url) {
  const [data, setData] = React.useState(null);
  const [loading, setLoading] = React.useState(true);
  React.useEffect(() => {
    fetch(url)
      .then(r => r.json())
      .then(d => { setData(d); setLoading(false); });
  }, [url]);
  return { data, loading };
}

// Context provider
```

```
const ThemeContext = React.createContext('light');
function ThemeProvider({ children }) {
  const [theme, setTheme] = React.useState('light');
  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

// Using React.memo
const Expensive = React.memo(function
Expensive({ value }) {
  // heavy calculation...
  return <div>{value}</div>;
});
```

--- End of Notes ---