React Concepts Explained

1. JSX (JavaScript XML)

- A syntax extension that lets you write HTML-like code in JavaScript.
- Makes UI code more readable.

```
const element = <h1>Hello React</h1>;
```

---

2. Components

- Building blocks of React apps.
- Two types:
  - **Class Components** (older, with lifecycle methods).
  - **Functional Components** (modern, with hooks).

---

3. Lifecycle Methods

- Special methods in **class components** that run at different stages:
  - `componentDidMount` → after component renders.
  - `componentDidUpdate` → after state/props update.
  - `componentWillUnmount` → before removal.
- In functional components → replaced by **`useEffect` hook**.

---

4. Props (Properties)

- Pass data from parent → child component.
- Immutable inside the child.

```
<Greeting name="Alice" />
```

---

5. Hooks

- Functions that let functional components use features like **state & lifecycle**.
- Important ones:
  - `useState` → state management.
  - `useEffect` → side effects.
  - `useContext` → global data access.

- `useReducer` → complex state management.

---

6. State

- Internal data that changes over time.

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

---

7. Virtual DOM

- React's lightweight copy of the real DOM.
- Compares changes (diffing) and updates only necessary parts → better performance.

---

8. Advanced State Management

- For large apps where state is shared deeply.
- Options:
  - **Context API** (built-in).
  - **Redux, Zustand, MobX, Recoil** (external libraries).

---

9. Conditional Rendering

- Show UI based on conditions.

```
{isLoggedIn ? <Dashboard /> : <Login />}
```

---

10. Context API

- Provides a way to share global data without **prop drilling**.

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

---

11. Higher-Order Components (HOCs)

- Functions that take a component and return a new component with extra features.

```
const withLogger = (WrappedComponent) => {
  return (props) => {
    console.log("Props:", props);
    return <WrappedComponent {...props} />;
  };
};
```

---

12. React Router

- For navigation between pages in a single-page app (SPA).

```
<Route path="/about" element={<About />} />
```

---

13. Redux (State Management Library)

- Centralized store for global state.
- Uses **actions**, **reducers**, and **dispatch**.

---

14. Building Your First React Component

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

---

15. Controlled Components

- Form inputs where **React state controls the value**.

```
<input type="text" value={name} onChange={e => setName(e.target.value)} />
```

---

16. Event Handling

- React events use **camelCase** and functions.

```
<button onClick={handleClick}>Click Me</button>
```

---

17. React Elements

- Smallest building blocks of React.
- Example:

```
const element = React.createElement('h1', null, 'Hello World');
```

---

18. React Hooks (again)

- Already covered above (core to functional components).

---

19. Stateless Functional Components

- Components with no state, just props.

- Example:

```
const Greeting = ({ name }) => <h1>Hello, {name}</h1>;
```

---

20. Lists & Keys

- Render lists using `map()`.
- Keys help React identify items.
- Example:

```
{items.map(item => <li key={item.id}>{item.name}</li>)}
```

---

21. Refs

- Access or manipulate DOM elements directly.
- Example:

```
const inputRef = useRef(null);
```

---

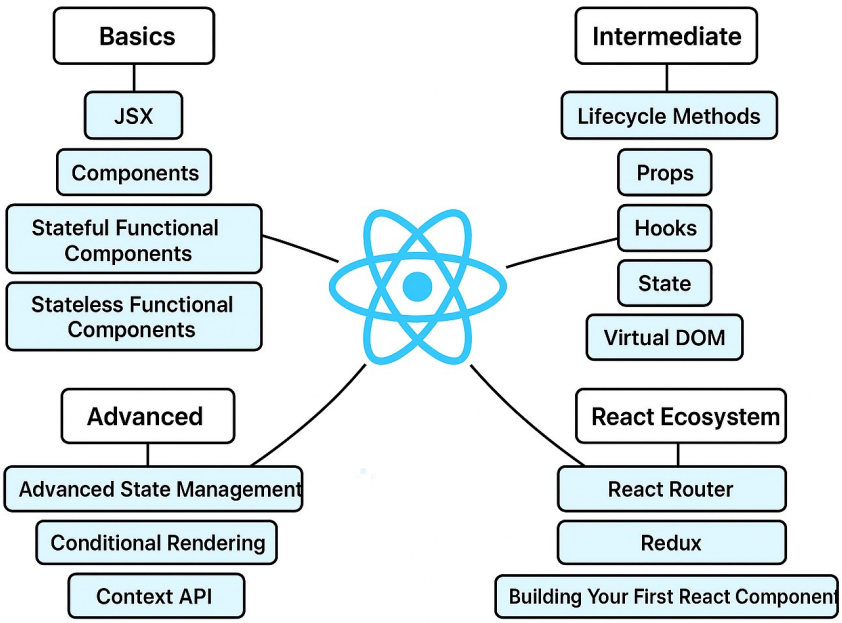
22. Error Boundaries

- Catch errors in component trees.

---

23. React Suspense & Lazy Loading

- Code splitting and loading components on demand.



In-depth Concepts:

1. React Fragments (<> </>)

- Lets you group multiple elements without adding extra nodes to the DOM.

```
return (
  <>
    <h1>Title</h1>
    <p>Description</p>
  </>
);
```

---

2. Rendering Lists

- Display data dynamically with JavaScript's `.map()` function.

```
const items = ["Apple", "Banana", "Cherry"];
return (
  <ul>
    {items.map((item, index) => <li key={index}>{item}</li>)}
  </ul>
);
```

---

3. Keys in Lists

- A **key** is a unique identifier React uses to optimize rendering.

```
<li key={user.id}>{user.name}</li>
```

---

4. Default Props

- Define fallback values when props aren't passed.

```
function Button({ text = "Click Me" }) {
  return <button>{text}</button>;
}
```

---

5. PropTypes (Type Checking)

- Helps catch bugs by checking the type of props.

```
import PropTypes from 'prop-types';

function Greeting({ name }) {
  return <h1>Hello {name}</h1>;
}

Greeting.propTypes = {
  name: PropTypes.string.isRequired,
};
```

---

6. Styling in React

- Multiple ways to style your components:
    - **Inline styles**: `<div style={{ color: "red" }}>Hi</div>`
    - **CSS modules**: Scoped CSS for components.
    - **Styled-components / Tailwind** (advanced but useful later).

---

7. Forms in React

- **Controlled components**: Inputs controlled by React state.

```
const [name, setName] = useState("");
return (
  <input
    type="text"
    value={name}
    onChange={e => setName(e.target.value)}
  />
);
```

---

8. Lifting State Up

- When two child components need shared data, move state up to the parent.

```
function Parent() {
  const [count, setCount] = useState(0);
  return (
    <>
      <Child1 count={count} />
      <Child2 setCount={setCount} />
    </>
  );
}
```

---

9. Conditional Class Names

- Dynamically apply classes.

```
<div className={isActive ? "active" : "inactive"}>Hello</div>
```

---

10. useEffect Basics

- For side effects (fetching data, timers, DOM changes).

```
useEffect(() => {
  console.log("Component mounted");
}, []); // runs once
```

11. React Developer Tools

- A browser extension to debug React apps (inspecting components, props, and state).