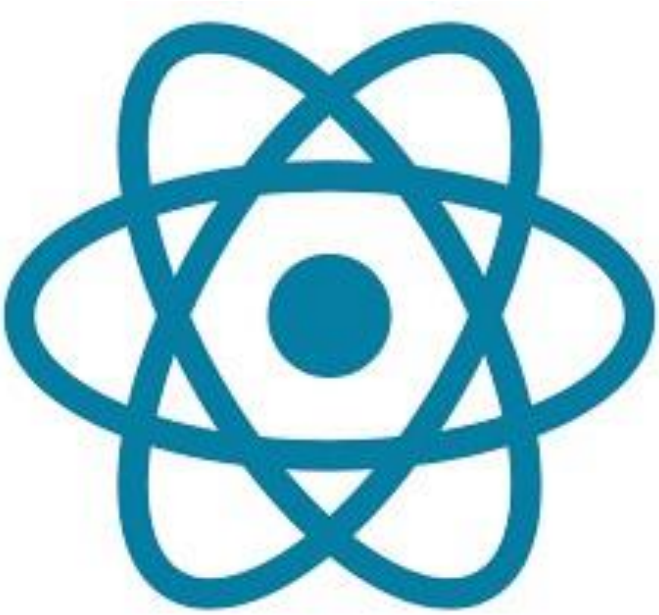

ReactJS Cheatsheet

React JS cheatsheet: key concepts, components, hooks, lifecycle methods.

1. **JSX Syntax:** JSX allows you to write HTML elements in JavaScript and place them in the DOM without using methods like `createElement`.

Code: const element = <h1>Hello, world!</h1>;

2. **Rendering Elements:** ReactDOM's `render` method is used to render React elements into the DOM.

Code: ReactDOM.render(element, document.getElementById('root'));

3. **Components:** Functional components are basic JavaScript functions that return React elements.

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

4. **Class Components:** Class components are ES6 classes that extend `React.Component` and must contain a `render` method returning React elements.

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

5. **Props:** Props are read-only attributes passed to components to allow data to flow from parent to child.

Code: const element = <Welcome name="Sara" />;

6. **State:** State is a special object in React components that determines the component's behavior and how it will render.

Code: class Clock extends React.Component { constructor(props) { super(props); this.state = {date: new Date()}; } }

7. **Lifecycle Methods:** Lifecycle methods are special methods in React components that run at different points in a component's life (e.g., `componentDidMount` runs after the component is mounted).

Code: componentDidMount() { this.timerID = setInterval(() => this.tick(), 1000); }

8. **Handling Events:** React events are named using camelCase, and you pass a function as the event handler rather than a string.

Code: <button onClick={this.handleClick}>Click me</button>

9. **Conditional Rendering:** Conditional rendering in React allows you to render different elements or components based on a condition.

Code: {isLoggedIn ? <LogoutButton /> : <LoginButton />}

10. **Lists and Keys:** Keys help React identify which items have changed, are added, or are removed and should be given to elements inside a loop.

```
const listItems = numbers.map((number) => <li key={number.toString()}>{number}</li>);
```

11. **Forms:** Controlled components are forms that have their input values controlled by the component's state.

Code: handleChange(event) { this.setState({value: event.target.value}); }

12. **Lifting State Up:** Lifting state up involves moving state to a common ancestor of components that need to share that state.

```
handleTemperatureChange(temperature) { this.setState({temperature}); }
```

13. **Composition vs Inheritance:** React uses composition over inheritance to allow components to be nested within each other and reused.

Code: function FancyBorder(props) { return <div className={'FancyBorder FancyBorder-' + props.color}>{props.children} ; }

14. **Hooks (useState):** The `useState` hook lets you add state to functional components.

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

15. **Hooks (useEffect):** The `useEffect` hook allows you to perform side effects in functional components.

Code: useEffect(() => { document.title = `You clicked ${count} times`; });

16. **Context:** The Context API is used to pass data through the component tree without having to pass props down manually at every level.

Code: const MyContext = React.createContext(defaultValue);

17. **Error Boundaries:** Error boundaries are React components that catch JavaScript errors anywhere in their child component tree.

Code: class ErrorBoundary extends React.Component { componentDidCatch(error, info) { logErrorToMyService(error, info); } }

18. **Refs:** Refs provide a way to access DOM nodes or React elements created in the `render` method.

Code: this.myRef = React.createRef();