React.js Cheatsheet

1 JSX Syntax

JSX allows HTML-like syntax in JavaScript. It gets compiled to `React.createElement`.

```
import React from 'react';

function App() {
  return <div>Hello, <strong>World</strong>!</div>;
}
```

2 Functional Components

Components are reusable UI pieces, defined as functions.

```
function Welcome({ name }) {
  return <h1>Hello, {name}!</h1>;
}

const App = () => <Welcome name="Alice" />;
```

3 Props and State

Props pass data to components; state manages dynamic data.

```
import { useState } from 'react';

function Counter(props) {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>{props.label}: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

4 Event Handling

Handle user interactions with event handlers.

```
function Button() {
  const handleClick = () => alert('Clicked!');
  return <button onClick={handleClick}>Click Me</button>;
}
```

5 Conditional Rendering

Render elements based on conditions.

```
function LoginStatus({ isLoggedIn }) {
  return isLoggedIn ? <p>Welcome!</p> : <p>Please log in.</p>;
}
```

6 Lists and Keys

Render lists with unique `key` prop for efficient updates.

```
function ItemList({ items }) {
  return (
    <ul>
      {items.map(item => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  );
}
```

7 useState Hook

Manage state in functional components.

```
import { useState } from 'react';

function Toggle() {
  const [isOn, setIsOn] = useState(false);
  return (
    <button onClick={() => setIsOn(!isOn)}>
      {isOn ? 'On' : 'Off'}
    </button>
  );
}
```

8 useEffect Hook

Handle side effects (e.g., data fetching, subscriptions).

```
import { useState, useEffect } from 'react';

function DataFetcher() {
  const [data, setData] = useState(null);
  useEffect(() => {
    fetch('https://api.example.com/data')
      .then(res => res.json())
      .then(setData);
    return () => console.log('Cleanup');
  }, []); // Empty array: run once
  return <div>{data ? data.name : 'Loading...'}</div>;
}
```

9 Basic Custom Hooks

Create reusable logic with custom hooks.

```
import { useState, useEffect } from 'react';

function useWindowSize() {
  const [size, setSize] = useState({ width: 0, height: 0 });
  useEffect(() => {
    const updateSize = () => setSize({
      width: window.innerWidth,
      height: window.innerHeight
    });
    window.addEventListener('resize', updateSize);
    updateSize();
    return () => window.removeEventListener('resize', updateSize);
  }, []);
  return size;
}

function Component() {
  const { width, height } = useWindowSize();
  return <p>Window: {width}x{height}</p>;
}
```

10 Context API

Share data across components without prop drilling.

```
import { createContext, useContext } from 'react';

const ThemeContext = createContext('light');

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return <button className={theme}>Click</button>;
}

function App() {
  return (
    <ThemeContext.Provider value="dark">
      <ThemedButton />
    </ThemeContext.Provider>
  );
}
```

11 Component Lifecycle

Manage component phases with hooks (functional components).

```
import { useEffect } from 'react';

function MyComponent() {
  useEffect(() => {
    console.log('Mounted');
    return () => console.log('Unmounted'); // Cleanup
  }, []); // Mount/Unmount
  useEffect(() => {
    console.log('Updated');
  }); // Every render
  return <div>Hello</div>;
}
```

12 Best Practices

- Use functional components and hooks over class components.
- Keep components small and focused.
- Always provide unique `key` props for lists.
- Avoid direct state mutations; use state setters.
- Clean up side effects in `useEffect` to prevent memory leaks.
- Use descriptive prop and variable names.