



REACT JS CHEAT SHEET



Muhammad Mohsin

SWIP LEFT
FOR MORE



Basic Concepts

JSX - JavaScript XML. Allows writing **HTML structures** in JavaScript files using **XML-like syntax**.

Components - Independent, reusable **pieces of UI**. Can be **functional** or **class-based**.

Props - Short for **properties**, these are **read-only inputs** to components that define **attributes** or **configuration**.

State - Holds data that might change over the **lifecycle** of a component. Used in **class components** and **functional components** via the **useState hook**.

Components

Functional components and class components are two ways to build components in React, each with its distinct characteristics:

- Class Components: Before the introduction of Hooks in **React 16.8**, this was the only way to **create components** with state and access **lifecycle methods**. They require using the `class` keyword to extend '**React.Component**' and offer a **more verbose syntax**.

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



Muhammad Mohsin

SWIP LEFT
FOR MORE



- Functional Components: Initially used for stateless components (**presentational components**), the introduction of **Hooks** has enabled the use of **local state, side effects** and **other React features**, making **functional components** almost universally preferred for their **concise syntax** and **ease of maintenance**.

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

While class components provide all React features via an object-oriented syntax, functional components with Hooks are now favored for their simplicity and expressiveness.

Creating Components

- **Functional Component with State:**

```
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>  
  );  
}
```



Muhammad Mohsin

SWIP LEFT
FOR MORE



- Class Component with State and Lifecycle Methods:

```
class Counter extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = { count: 0 };  
  }  
  
  componentDidMount() {  
    // ComponentDidMount: Code to run after component mounts  
  }  
  
  componentDidUpdate() {  
    // componentDidUpdate: Code to run after updating occurs  
  }  
  
  componentWillUnmount() {  
    // componentWillUnmount: Cleanup before component unmounts  
  }  
  
  render() {  
    return (  
      <div>  
        <p>You clicked {this.state.count} times</p>  
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>  
          Click me  
        </button>  
      </div>  
    );  
  }  
}
```

Hooks

- useState



```
const [state, setState] = useState(initialState);
```



Muhammad Mohsin

SWIPE LEFT
FOR MORE



- **useEffect**

```
useEffect(() => {
  // Side effects here
  return () => {
    // Cleanup (optional)
  };
}, [dependencies]);
```

- **useContext**



```
const value = useContext(MyContext);
```

- **useReducer**



```
const [state, dispatch] = useReducer(reducer, initialArg, init);
```

- **useCallback**



```
const memoizedCallback = useCallback(() => {
  // Your callback function
}, [dependencies]);
```



Muhammad Mohsin

SWIPE LEFT
FOR MORE



- **useMemo**



```
const memoizedValue = useMemo(() => computeExpensiveValue(a, b), [a, b]);
```

- **useRef**



```
const myRef = useRef(initialValue);
```

- **useTransition**



```
const [isPending, startTransition] = useTransition();
```

Conditional Rendering

Inline If with Logical && Operator:

```
{condition && <Component />}
```

Inline If-Else with Conditional Operator:

```
{condition ? <Component1 /> : <Component2 />}
```



Muhammad Mohsin

SWIPE LEFT
FOR MORE



Lists and Keys

Rendering Multiple Components



```
{data.map((item) => <Component key={item.id} item={item} />)}
```

Handling Events



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

Lifting State Up

Sharing state between components by moving it to their closest common ancestor.

Context API

Used to pass data through the component tree without having to pass props down manually at every level.

Fragments

Used to group a list of children without adding extra nodes to the DOM.

```
<React.Fragment>  
  <ChildA />  
  <ChildB />  
</React.Fragment>
```



Muhammad Mohsin

SWIPE LEFT
FOR MORE



Higher-Order Components (HOC)

A function that takes a component and returns a new component, used for reusing component logic.

Forwarding Refs

Used to pass refs down to child components.

```
const FancyButton = React.forwardRef((props, ref) => (
  <button ref={ref} {...props}>
    {props.children}
  </button>
));
```

Concurrent Features in React 18 and beyond

- Automatic batching: React 18 automatically batches more state updates.
- Suspense: Lets your components "wait" for something before they can render, making it easier to split code and manage loading states.
- useDeferredValue, useTransition: For managing transitions and prioritizing resource loading.

This **cheatsheet** covers **foundational concepts** and **common hooks** in **React development**.

Remember, **React** and its **ecosystem** are vast, and continuous **learning** and **practice** are key to **mastering it**.



Muhammad Mohsin

SWIPE LEFT
FOR MORE





Follow
Me For More Content

Muhammad Mohsin