React JS Cheatsheet for Interview Preparation

For Freshers and Experienced Developers
A simple, detailed guide to explain React JS concepts in interviews with examples.

1. What is React JS and Why Use It?

What is it? React JS is a JavaScript library by Facebook for building interactive web interfaces. Think of it as LEGO blocks: you create small pieces (components) like buttons or forms and combine them to build apps.

Why Use It?

- **Fast:** Updates only changed parts of the page using Virtual DOM.
- **Reusable:** Build components once, use them anywhere.
- **Community:** Tons of tools (e.g., Redux, Next.js) and job opportunities.
- **Flexible:** Works for web, mobile (React Native), and server-side rendering.

Interview Tip: Say, "React is great for building fast, reusable UIs. It's like LEGO—you make small components and snap them together."

2. Virtual DOM

What is it? A lightweight copy of the real DOM (the web page's structure) stored as a JavaScript object.

How It Works:

1. When something changes (e.g., clicking a button), React makes a new Virtual DOM.
2. It compares the new and old Virtual DOMs to find differences (called diffing).
3. Only the differences update the real DOM, making it fast.

Example: If a list has 100 items and one changes, React updates just that one item.

Interview Tip: Explain, "It's like editing a draft document before updating the final version to save time."

3. Reconciliation

What is it? The process where React updates the real DOM by comparing old and new component trees.

How It Works: React checks what changed (diffing) and applies minimal updates. Using `key` attributes in lists helps it know which items changed.

Example:

```
<ul>
  <li key="1">Apple</li>
  <li key="2">Banana</li>
</ul>
```

Adding a new item updates only that item, not the whole list.

Interview Tip: Say, "Reconciliation is like updating only the changed parts of a puzzle, not rebuilding it."

4. React Fiber

What is it? React's updated engine (since React 16) for smoother updates.

How It Works:

- Splits work into small chunks (fibers).
- Can pause low-priority tasks (e.g., background loading) to focus on urgent ones (e.g., button clicks).
- Makes apps feel faster, especially for animations.

Example: A button click updates instantly while a big list loads in the background.

Interview Tip: Explain, "Fiber is like a smart manager who prioritizes important tasks to keep the app responsive."

5. State

What is it? Data that changes in a component, like a counter or form input, causing the UI to update.

Example:

```
import { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Add</button>
    </div>
  );
}
```

Interview Tip: Say, "State is like a component's memory—it holds data that can change and updates the UI."

6. Props

What is it? Data passed from a parent component to a child, like passing notes.

Example:

```
function Greeting({ name }) {
  return <h1>Hello, {name}!</h1>;
}
function App() {
  return <Greeting name="Alice" />;
}
```

Interview Tip: Explain, "Props are like instructions a parent gives to a child component, and they're read-only."

7. Props Drilling

What is it? Passing props through many nested components to reach a deep child.

Example:

```
function App() {
  return <Parent name="Alice" />;
}
function Parent({ name }) {
  return <Child name={name} />;
}
function Child({ name }) {
  return <h1>Hello, {name}!</h1>;
}
```

Issue: Gets messy with many layers.

Interview Tip: Say, "It's like passing a note through multiple people—Context or Redux avoids this."

8. useState Hook

What is it? A way to add state to functional components.

How It Works: Gives you a state variable and a function to update it. Updates trigger re-renders.

Example:

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

Interview Tip: Explain, “useState is like a light switch—it tracks state and flips the UI when it changes.”

9. useEffect Hook

What is it? A hook for side effects like fetching data or setting timers.

How It Works: Runs after rendering. Use a dependency array to control when it runs.

Example:

```
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');
  }, []); // Runs once
  return <div>{data ? data.name : 'Loading...'}</div>;
}
```

Interview Tip: Say, “useEffect is like setting up a task (e.g., fetching data) after the component appears.”

10. useRef Hook

What is it? A hook to hold values that don’t trigger re-renders, like a DOM reference.

How It Works: Returns an object with a `.current` property you can change.

Example:

```
import { useRef } from 'react';
function TextInput() {
  const inputRef = useRef(null);
  const focusInput = () => inputRef.current.focus();
  return (
    <div>
      <input ref={inputRef} />
      <button onClick={focusInput}>Focus</button>
    </div>
  );
}
```

Interview Tip: Explain, “useRef is like a sticky note you attach to a DOM element or value.”

11. useLayoutEffect Hook

What is it? Like `useEffect`, but runs before the browser paints the screen.

How It Works: Good for DOM measurements (e.g., element size).

Example:

```
import { useLayoutEffect, useRef } from 'react';
function Box() {
  const divRef = useRef(null);
  useLayoutEffect(() => {
    const { height } = divRef.current.getBoundingClientRect();
    console.log('Height:', height);
  }, []);
  return <div ref={divRef} style={{ height: '100px' }}>Box</div>;
}
```

Interview Tip: Say, "useLayoutEffect is like useEffect but runs earlier to adjust the UI before it shows."

12. useReducer Hook

What is it? A hook for complex state logic, like a mini-Redux.

How It Works: Uses a reducer function to update state based on actions.

Example:

```
import { useReducer } from 'react';
const initialState = { count: 0 };
function reducer(state, action) {
  switch (action.type) {
    case 'increment': return { count: state.count + 1 };
    case 'decrement': return { count: state.count - 1 };
    default: return state;
  }
}
function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>+</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>-</button>
    </div>
  );
}
```

Interview Tip: Explain, "useReducer is like a traffic controller for complex state changes."

13. useContext Hook

What is it? A hook to share data across components without props drilling.

Example:

```
import { createContext, useContext } from 'react';
const ThemeContext = createContext('light');
function App() {
  return (
    <ThemeContext.Provider value="dark">
      <Display />
```

```
    </ThemeContext.Provider>
  );
}
function Display() {
  const theme = useContext(ThemeContext);
  return <div>Theme: {theme}</div>;
}
```

Interview Tip: Say, "useContext is like a shared bulletin board all components can read from."

14. Lifecycle Methods with useEffect

What is it? In class components, methods like `componentDidMount` handle lifecycle events. In functional components, `useEffect` covers them.

How It Works:

- **Mount:** `useEffect(() => , [])` runs once after component mounts.
- **Update:** `useEffect(() => , [dep])` runs when dependencies change.
- **Unmount:** Return a cleanup function in `useEffect`.

Example:

```
import { useEffect } from 'react';
function Component() {
  useEffect(() => {
    console.log('Mounted');
    return () => console.log('Unmounted');
  }, []);
  return <div>Hello</div>;
}
```

Interview Tip: Explain, "useEffect handles a component's life stages, like birth, updates, and cleanup."

15. useMemo Hook

What is it? A hook to cache expensive calculations to avoid repeating them.

How It Works: Only recalculates if dependencies change.

Example:

```
import { useMemo, useState } from 'react';
function ExpensiveComponent({ numbers }) {
  const [count, setCount] = useState(0);
  const sum = useMemo(() => {
    console.log('Calculating...');
    return numbers.reduce((a, b) => a + b, 0);
  }, [numbers]);
  return (
    <div>
      <p>Sum: {sum}</p>
      <button onClick={() => setCount(count + 1)}>Re-render</button>
```

```
    </div>
  );
}
```

Interview Tip: Say, “useMemo is like saving a math result to avoid recalculating it.”

16. useCallback Hook

What is it? A hook to cache functions to avoid recreating them on re-renders.

How It Works: Only recreates the function if dependencies change.

Example:

```
import { useCallback, useState } from 'react';
function Button({ onClick }) {
  return <button onClick={onClick}>Click</button>;
}
function App() {
  const [count, setCount] = useState(0);
  const handleClick = useCallback(() => setCount(count + 1), [count]);
  return <Button onClick={handleClick} />;
}
```

Interview Tip: Explain, “useCallback is like saving a recipe so you don’t rewrite it every time.”

17. Optimizing Performance in React

How to Optimize:

- **useMemo:** Cache calculations.
- **useCallback:** Cache functions.
- **React.memo:** Prevent re-renders of unchanged components.
- **Lazy Loading:** Use `React.lazy` for big components.
- **Keys:** Use keys in lists to avoid re-rendering unchanged items.

Example:

```
import { memo, useMemo, useCallback } from 'react';
const Child = memo(({ data, onClick }) => {
  console.log('Child rendered');
  return <button onClick={onClick}>{data}</button>;
});
function App() {
  const [count, setCount] = useState(0);
  const data = useMemo(() => 'Hello', []);
  const handleClick = useCallback(() => setCount(count + 1), [count]);
  return <Child data={data} onClick={handleClick} />;
}
```

Interview Tip: Say, “Optimization is like tuning a car—useMemo and useCallback make it run efficiently.”

18. Pros and Cons of React JS

Pros:

- Fast updates with Virtual DOM.
- Reusable components save time.
- Huge ecosystem (React Native, Next.js).
- Flexible with other tools.

Cons:

- Takes time to learn (especially hooks).
- Needs extra libraries for routing or state management.
- JSX can feel strange at first.

Interview Tip: Say, "React is powerful and flexible but needs extra tools for big apps."

19. Redux and Global State

What is Redux? A library to manage app-wide data (global state) in one place.

What is Global State? Data shared across components, like a user's login status.

How Redux Works:

- **Store:** One place for all app data.
- **Actions:** Instructions to change data (e.g., `{ type: 'ADD' }`).
- **Reducers:** Rules to update the store based on actions.

Interview Tip: Explain, "Redux is like a central bank—everyone accesses the same account."

20. Global State vs. Props Drilling

Global State (Redux/Context):

- Shares data without passing through components.
- Cleaner for big apps.

Props Drilling:

- Passes data through every component layer.
- Okay for small apps, messy for big ones.

Example (Context vs. Drilling):

```
import { createContext, useContext } from 'react';
const UserContext = createContext(null);
function App() {
  return (
    <UserContext.Provider value="Alice">
      <Child />
```

```
    </UserContext.Provider>
  );
}
function Child() {
  const user = useContext(UserContext);
  return <p>User: {user}</p>;
}
```

Interview Tip: Say, “Global state skips the hassle of passing props through many layers.”

21. Redux Toolkit Example

What is it? A simpler way to use Redux with less code.

Example:

```
import { configureStore, createSlice } from '@reduxjs/toolkit';
import { Provider, useDispatch, useSelector } from 'react-redux';

// Create slice
const counterSlice = createSlice({
  name: 'counter',
  initialState: { count: 0 },
  reducers: {
    increment(state) { state.count += 1; },
    decrement(state) { state.count -= 1; },
  },
});
const { increment, decrement } = counterSlice.actions;

// Create store
const store = configureStore({
  reducer: { counter: counterSlice.reducer },
});

// Component
function Counter() {
  const count = useSelector(state => state.counter.count);
  const dispatch = useDispatch();
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => dispatch(increment())}>+</button>
      <button onClick={() => dispatch(decrement())}>-</button>
    </div>
  );
}

// App
function App() {
  return (
    <Provider store={store}>
      <Counter />
    </Provider>
  );
}
```

Interview Tip: Explain, “Redux Toolkit is like Redux with training wheels—less boilerplate.”

22. Interview Tips

- **Freshers:** Focus on `useState`, `useEffect`, props, and Context.
- **Pros:** Know `useMemo`, `useCallback`, Redux, and optimization.
- **Explain with Analogies:** Use LEGO for components, drafts for Virtual DOM.
- **Show Examples:** Reference your projects or the code above.

React Docs for more details.