

React Quick Reference

Components, hooks, state, effects, patterns

JSX Basics

Expressions & Attributes

```
const name = "Alice";
const el = <h1>Hello, {name}!</h1>;
const img = <img src={url} alt="photo" />;
```

JSX Rules

{expression}	Embed any JS expression
className	Use instead of class
htmlFor	Use instead of for
style={{color: 'red'}}	Inline styles as object
<Component />	Self-closing tags required
<> ... </>	Fragment (no extra DOM node)

Components

Function Components

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

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

Props

```
function Card({ title, children }) {
  return (
    <div className="card">
      <h2>{title}</h2>
      {children}
    </div>
  );
}

<Card title="Welcome">
  <p>Content here</p>
</Card>
```

Default Props

```
function Button({ label = "Click me", onClick }) {
  return <button onClick={onClick}>{label}</button>;
}
```

State (useState)

Basic State

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

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

Functional Updates

```
setCount(prev => prev + 1); // use prev state
setItems(prev => [...prev, newItem]); // arrays
setUser(prev => ({...prev, name: "Bob"})); // objects
```

State Rules

Immutable updates	Never mutate state directly
Async batching	Updates may be batched
Functional form	Use prev => when depending on prior state

Effects (useEffect)

Effect Patterns

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

// Run on every render
useEffect(() => { /* ... */ });

// Run once on mount
useEffect(() => { /* ... */ }, []);

// Run when deps change
useEffect(() => { /* ... */ }, [count]);

// Cleanup on unmount
useEffect(() => {
  const id = setInterval(tick, 1000);
  return () => clearInterval(id);
}, []);
```

Lists & Keys

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

Keys must be stable, unique IDs -- avoid array index as key

Event Handling

Events

```
<button onClick={handleClick}>Click</button>
<button onClick={() => handleDelete(id)}>Del</button>
<input onChange={(e) => setVal(e.target.value)} />
<form onSubmit={(e) => {
  e.preventDefault();
  handleSubmit();
}}>
```

Common Events

onClick	Mouse click
onChange	Input value change
onSubmit	Form submission
onKeyDown	Key press
onFocus / onBlur	Focus gained / lost
onMouseEnter	Mouse enters element

Conditional Rendering

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

// Logical AND (short-circuit)
{hasError && <ErrorBanner />}

// Early return
function Page({ user }) {
  if (!user) return <Login />;
  return <Dashboard user={user} />;
}
```

Hooks

useRef

```
const inputRef = useRef(null);
// Access: inputRef.current.focus();
<input ref={inputRef} />
```

useRef persists values across renders without triggering re-render

useMemo & useCallback

```
// Memoize expensive computation
const sorted = useMemo(
  () => items.sort(compareFn),
  [items]
);

// Memoize callback
const handleClick = useCallback(
  () => setCount(c => c + 1),
  []
);
```

useContext

```
import { useContext } from "react";
const value = useContext(MyContext);
```

Custom Hooks

```
function useLocalStorage(key, initial) {
  const [value, setValue] = useState(() => {
    const saved = localStorage.getItem(key);
    return saved ? JSON.parse(saved) : initial;
  });

  useEffect(() => {
    localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
}

// Usage
const [name, setName] = useLocalStorage("name", "");
```

Custom hooks must start with 'use'

Context API

Create & Provide

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

const ThemeCtx = createContext("light");

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

Consume

```
function Page() {
  const theme = useContext(ThemeCtx); // "dark"
  return <div className={theme}>...</div>;
}
```