REACT QUICK REFERENCE

JSX Basics

Expressions & Attributes

```
const name = "Alice";
const el =
Hello, {name}!
;
const img = 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
Hello, {name}!
;
}

// Arrow function variant
const Greeting = ({ name }) => (

Hello, {name}!

);
```

Props

```
function Card({ title, children }) {
  return (


{title}

      {children}


  );
}



Content here
```

Default Props

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

State (useState)

Basic State

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

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

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);
}, []);
```

REACT QUICK REFERENCE (continued)

Lists & Keys

```
function TodoList({ items }) {
  return (


      {items.map(item => (
• {item.text}
      ))}


  );
}
```

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

Event Handling

Events

```
Click
handleDelete(id)}>Del
 setVal(e.target.value)}
/>
 {
  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 ?  : }

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

// Early return
function Page({ user }) {
  if (!user) return ;
  return ;
}
```

Hooks

useRef

```
const inputRef = useRef(null);
// Access: inputRef.current.focus();
```

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 (



  );
}
```

Consume

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