React クイックリファレンス

コンポーネント、フック、状態、エフェクト、パターン

JSX の基本

式と属性

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

JSX のルール

{expression}	任意の JS 式を埋め込む
className	**class** の代わりに使用
htmlFor	**for** の代わりに使用
style={{color: 'red'}}	インラインスタイルをオブジェクトで指定
<Component />	自己閉じタグが必須
<> ... </>	フラグメント（余分な DOM ノードなし）

コンポーネント

関数コンポーネント

```
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>
```

デフォルト Props

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

状態（useState）

基本的な状態

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

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

関数型の更新

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

状態のルール

Immutable updates	状態を直接変更しない
Async batching	更新はバッチ処理される場合がある
Functional form	前の状態に依存する場合は **prev =>**を使用

エフェクト（useEffect）

エフェクトのパターン

```
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);
}, []);
```

リストとキー

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

キーは安定したユニーク ID にする — 配列インデックスはキーとして使わない

イベント処理

イベント

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

よく使うイベント

onClick	マウスクリック
onChange	入力値の変更
onSubmit	フォームの送信
onKeyDown	キー押下
onFocus / onBlur	フォーカス取得 / 喪失
onMouseEnter	マウスが要素に入る

条件付きレンダリング

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

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

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

フック

useRef

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

useRef は再レンダリングをトリガーせずにレンダリング間で値を保持する

useMemo と 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);
```

カスタムフック

```
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", "");
```

カスタムフックは'use'で始まる名前にする必要がある

Context API

作成とプロバイダー

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

const ThemeCtx = createContext("light");

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

消費

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