React 빠른 참조

컴포넌트, 훅, 상태, 이펙트, 패턴

JSX 기본

표현식 및 속성

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

JSX 규칙

{expression}	모든 JS 표현식 삽입
className	**class** 대신 사용
htmlFor	**for** 대신 사용
style={{color: 'red'}}	인라인 스타일을 객체로
<Component />	자기 닫힘 태그 필수
<> ... </>	프래그먼트 (추가 DOM 노드 없음)

컴포넌트

함수 컴포넌트

```
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>
```

기본 Props

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

상태 (useState)

기본 상태

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

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

함수형 업데이트

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

상태 규칙

불변 업데이트	상태를 직접 변경하지 말 것
비동기 배치	업데이트는 배치될 수 있음
함수형 형태	이전 상태에 의존할 때 **prev =>** 사용

이펙트 (useEffect)

이펙트 패턴

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

목록 및 키

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

키는 안정적이고 고유한 ID여야 함 — 배열 인덱스를 키로 사용하지 말 것

이벤트 처리

이벤트

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

일반 이벤트

onClick	마우스 클릭
onChange	입력 값 변경
onSubmit	폼 제출
onKeyDown	키 누름
onFocus / onBlur	포커스 획득 / 상실
onMouseEnter	마우스가 요소에 진입

조건부 렌더링

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

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

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

훅

useRef

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

useRef는 리렌더링 없이 렌더 간 값을 유지

useMemo 및 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);
```

커스텀 훅

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

커스텀 훅은 반드시 'use'로 시작해야 함

Context API

생성 및 제공

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

const ThemeCtx = createContext("light");

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

소비

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