

컴포넌트와 JSX (Components & JSX)

React는 UI를 독립적이고 재사용 가능한 컴포넌트로 분리하여 구축하는 선언형 라이브러리입니다.

함수형 컴포넌트 및 Props

```
function Greeting({ name = "게스트", isOnline }) {
  return (
    <div className="user-card">
      <h1>안녕하세요, {name}님!</h1>
      {isOnline ? <p>온라인 상태</p> : <p>오프라인</p>}
    </div>
  );
}
```

리스트 렌더링 및 프래그먼트

```
const items = [{ id: 1, text: "사과" }, { id: 2, text: "바나나" }];

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

핵심 리액트 훅 (Core Hooks)

useState (상태 관리)

```
const [count, setCount] = useState(0);
// 함수형 업데이트 (이전 상태 기반)
setCount((prev) => prev + 1);
```

useEffect (부수 효과 처리)

```
useEffect(() => {
  const timer = setInterval(() =>
    console.log('Tick'), 1000);
  return () => clearInterval(timer); // 클린업 함수
}, [dependency]); // 의존성 배열
```

useRef (DOM 접근 및 변경 불가능한 값 저장)

```
const inputRef = useRef(null);
const focusInput = () =>
  inputRef.current.focus();
```

useContext (전역 컨텍스트 구독)

```
const ThemeContext =
  createContext('light');
function ThemedButton() {
  const theme =
    useContext(ThemeContext);
  return <button className={theme}>버튼
</button>;
}
```

useReducer (복잡한 상태 로직 관리)

```
function reducer(state, action) {
  switch (action.type) {
    case 'increment': return { count:
      state.count + 1 };
    default: return state;
  }
}
const [state, dispatch] =
  useReducer(reducer, { count: 0 });
// dispatch({ type: 'increment' })
```

성능 최적화 훅 (Optimization)

- **useMemo**: 연산 비용이 큰 계산 결과 캐싱
const expensiveResult = useMemo(() => compute(data), [data]);
- **useCallback**: 컴포넌트 재생성 시 함수 인스턴스 유지
const handleClick = useCallback(() => { doSomething(id); }, [id]);
- **React.memo**: Props가 변경되지 않았을 때 리렌더링 방지
const PureItem = React.memo(function Item({ title }) { ... });

모던 React (Concurrent & Actions)

- **useTransition**: 긴급하지 않은 상태 업데이트를 백그라운드로 지연

```
const [isPending, startTransition] =
  useTransition();
startTransition(() =>
  setSearchQuery(input));
• useDeferredValue: 값의 업데이트를 지연시켜 UI 반응성 향상
const deferredQuery =
  useDeferredValue(searchQuery);
• Suspense & React.lazy: 코드 분할 및 비동기 로딩
const HeavyChart = React.lazy(() =>
  import('./HeavyChart'));
<Suspense fallback={<Spinner />}>
  <HeavyChart />
</Suspense>
```

이벤트 핸들링 및 폼 (Events & Forms)

- **합성 이벤트 (Synthetic Event)**: **onClick**, **onChange**, **onSubmit**
- **폼 제출 제어**:
const handleSubmit = (e) => {
 e.preventDefault(); // 기본 새로고침 방지
 const formData = new
 FormData(e.target);
};