React **CheatSheet**

React 치트 시트 (React Hooks 중심)

WITH
YAMOO9
DERESA KIM

React CheatSheet 2020

⚡ 핵심 개념 (Core Concepts)

⚡ React 훅 (Hooks)

⚡ 훅 고급 활용 (Advanced Hooks)

WITH
YAMOO9
DERESA KIM

핵심 개념

🎯 React 요소와 JS(X) → 요소 작성 기본 구문

```
// JSX를 사용하면 JavaScript에서 HTML을 작성할 수 있습니다.
// 웹 표준 HTML 요소(예: <div>, <span>, <h1> ~ <h6>, <form>, <input/> 등)를 사용할 수 있습니다.
<div>Hello React</div>
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎯 React 요소와 JS(X) → JSX는 표현 식(Expression)

```
// JSX는 표현 식(Expression)으로 변수에 할당 할 수 있습니다.
const greeting = <div>안녕! React</div>;

// 또는 조건에 따라 렌더링을 달리 할 수 있습니다.
const isNewToReact = true;

function sayGreeting() {
  if (isNewToReact) {
    return greeting;
  } else {
    return <div>React 멋짐!</div>
  }
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎯 React 요소와 JS(X) → JSX를 사용하면 표현 식(Expression) 중첩 가능

```jsx
const year = 2020;

// 중괄호({}) 안에 JS 표현 식을 삽입 할 수 있습니다.
// ※ 객체를 삽입 하려고 하면 오류가 발생합니다.
const greeting = <div>안녕! React 치트 시트 {year}</div>;
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎯 React 요소와 JS(X) → JSX를 사용하면 React 요소 중첩 가능

```
// 여러 줄에 JSX를 작성하려면 괄호 ()로 묶습니다.
const greeting = (
  // '부모' 또는 '자식' 요소는 HTML에서와 마찬가지로
  // JSX 요소를 서로 연관하여 설명하는 방법입니다.

  // <div>는 부모 요소입니다.
  <div>
    {/* <h1>과 <p>는 자식 요소입니다. */}
    <h1>안녕!</h1>
    <p>React를 겪하게 환영해!</p>
  </div>
);
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎯 React 요소와 JS(X) → HTML과 다소 다른 JSX 문법

```
// 내용을 포함하지 않는 HTML 요소는 <input> (HTML)이 아닌, <input /> (JSX)로 작성해야 합니다.
<input name="email" />

// 속성 이름은 유효한 JS 변수 이름 작성 규칙을 따라야 하며, camelCase 방식으로 작성합니다.
// 'class' (HTML) 대신, 'className'을 사용해야 합니다.
<button className="button is-save">저장</button>

// 내용을 포함하지 않는 React 요소는 닫는 태그(</ReactElement>)를 사용하지 않아도 됩니다.
<ReactElement />
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎯 React 요소와 JS(X) → React Web 렌더링에 필요한 3가지 (ReactDOM, JSX, DOM)

```
// NPM을 사용할 경우, import 구문을 사용합니다.
import React from "react";
import ReactDOM from "react-dom";

const greeting = <h1>안녕! React</h1>;

// ReactDOM.render(React 요소, 실제 DOM 요소 노드)
ReactDOM.render(greeting, document.getElementById("app"));
```

핵심 개념

🎛 컴포넌트 & props → React 컴포넌트 타입

```jsx
import React from "react";

// 컴포넌트 타입: 함수
function Header() {
  // 일반 JS 함수와 달리 함수 컴포넌트 이름으
  return <h1>안녕! React</h1>;
}

// ES6 화살표 함수로 작성할 수도 있습니다.
const Header = () => <h1>안녕! React
```

```jsx
import React from "react";

// 컴포넌트 타입: 클래스
class Header extends React.Component {
  // 클래스 컴포넌트는 함수 컴포넌트 보다 많은 상용구(boilerplate)를 사용합니다.
  // (extends 및 render 메서드 등)
  render() {
    return <h1>안녕! React</h1>;
  }
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎛️ 컴포넌트 & props → React 컴포넌트를 사용하는 방법

```
// 함수 컴포넌트를 일반 JS 함수처럼 호출하지 않습니다.
// JSX를 사용해 함수의 결과를 반환(return) 합니다.
const Header = () ⇒ <h1>안녕! React</h1>;

// 그리고 커스텀 React 요소(JSX)로 사용합니다.
ReactDOM.render(<Header />, document.getElementById("app"));
// 렌더링: <h1>안녕! React</h1>
```

핵심 개념

🧱 컴포넌트 & props → 앱에서 페이지 별, 컴포넌트 재 사용

```
// '/' 경로(route)에서 표시되는 페이지 컴포넌트
function IndexPage() {
  return (
    <div>
      <Header />
      <Hero />
      <Footer />
    </div>
  );
}
```

```
// '/about' 경로에서 표시되는 페이지 컴포넌트
function AboutPage() {
  return (
    <div>
      <Header />
      <About />
      <Testimonials />
      <Footer />
    </div>
  );
}
```

핵심 개념

🎛 컴포넌트 & props → 컴포넌트에 데이터를 전달할 경우, 속성(props) 사용

```
const username = "야무";

// props 라는 커스텀 '속성'을 추가 합니다.
ReactDOM.render(
  <Header username={username} />,
  document.getElementById("app")
);

// props는 모든 React 컴포넌트가 인자로 전달 받는 객체입니다.
function Header(props) {
  // 컴포넌트의 React 요소에 전달된 속성은
  // 컴포넌트 props 객체의 속성이 됩니다.
  return <h1>안녕! {props.username}</h1>;
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎚️ 컴포넌트 & props → 컴포넌트 속성(props)은 읽기 전용!

```
// 컴포넌트는 이상적으로 순수한 함수 여야 합니다.
// 즉, 모든 입력에 대한 동일한 출력이 기대되어야 합니다.

// props로는 다음을 수행할 수 없습니다.
function Header(props) {
  // 컴포넌트는 전달 받은 props 객체를 변경할 수 없으며 읽을 수만 있습니다.
  // ※ 전달 받은 데이터를 수정하려면? state를 사용해야 합니다.
  props.username = "데레사"; // ❌
  return <h1>안녕 {props.username}</h1>;
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎚️ 컴포넌트 & props → 칠드런(children)을 사용해 컴포넌트를 전달

```
// React 요소(또는 컴포넌트)를 props로 전달할 수 있습니다.
// props.children 이라는 특수한 속성으로 사용 됩니다.
function Layout(props) {
  return <div className="container">
}
```

```
// props.children을 통해 동일한 컴포넌트(예: Layout 컴포넌트)를
// 다른 컴포넌트(예: 페이지 컴포넌트) 간 공유할 수 있어 유용합니다.
function IndexPage() {
  return (
    <Layout>
      <Header />
      <Hero />
      <Footer />
    </Layout>
  );
}
```

```
function AboutPage() {
  return (
    <Layout>
      <About />
      <Footer />
    </Layout>
  );
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎛️ 컴포넌트 & props → JSX 안에서 3항 연산 식을 사용한 조건 처리

```jsx
// if 문을 사용해 조건 부로 렌더링을 처리할 수도 있지만
// 3항 연산자 식을 사용하면 JSX에서 손쉽게 조건 처리할 수 있습니다.
function Header() {
  const isAuth = checkAuth();
  return (
    <nav>
      <Logo />
      {/* isAuth 값이 true일 경우 AuthLinks 컴포넌트를 표시하고, 아닐 경우 Login 표시 */}
      {isAuth ? <AuthLinks /> : <Login />}
      {/* isAuth 값이 true인 경우 Greeting 컴포넌트 표시 */}
      {isAuth && <Greeting />}
    </nav>
  );
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🎛️ 컴포넌트 & props → React.Fragment 활용 (2개 이상 컴포넌트를 묶어야 할 경우)

```jsx
// 앞서 다룬 isAuth 조건 처리 구문을 개선할 수 있습니다.
function Header() {
  const isAuth = checkAuth();
  return (
    <nav>
      <Logo />
      {/* 2개 이상의 컴포넌트를 처리할 경우, <React.Fragment> 또는 <></>로 렌더링 처리해야 합니다. */}
      {isAuth ? (
        <>
          <AuthLinks />
          <Greeting />
        </>
      ) : <Login />}
    </nav>
  );
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🔑 리스트 & 키(Key) → 배열 순환 처리에는 map() 메서드 활용

```
const people = ["야무", "데레사", "지호"];
const peopleList = people.map(person ⇒ <p>{person}</p>);
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🔑 리스트 & 키(Key) → map() 메서드는 React 컴포넌트에서도 활용!

```
function App() {
  const people = ["야무", "데레사", "지호"];
  return (
    <ul>
      {/* {}를 사용해 반환 된 React 요소 집합을 보간 처리할 수 있습니다. */}
      {people.map(person => <Person name={person} />)}
    </ul>
  );
}

function Person({ name }) {
  // props에서 구조 분해 할당으로 'name' 속성을 추출하여 사용합니다.
  return <p>사용자 이름: {name}</p>;
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🔑 리스트 & 키(Key) → map() 메서드를 사용할 경우, key 속성은 필수!

```
function App() {
  const people = ["야무", "데레사", "지호"];
  return (
    <ul>
      {/* 키(key)는 고유한 ID 값을 가져야 합니다. */}
      {people.map(person ⇒ <Person key={person} name={person} />)}
    </ul>
  );
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🔑 리스트 & 키(Key) → key 속성 값은 고유한 ID를 사용!

```
// 데이터 집합에 고유한 식별자인 ID가 포함되지 않은 경우,
// map()의 2번째 매개 변수를 사용하여 각 요소 인덱스를 가져와 사용할 수 있습니다.
function App() {
  const people = ["야무", "데레사", "야무"];
  return (
    <ul>
      {people.map((person, i) => <Person key={`${person}-{i}`} name={person} />)}
    </ul>
  );
}
```

WITH
YAMOO9
DERESA KIM

핵심 개념

🔑 이벤트 & 이벤트 핸들러 → React 이벤트 속성 이름 (HTML과 다름)

```
// 참고: 대부분 이벤트 핸들러 함수는 'handle' 접두사로 시작합니다.
function handleToggleTheme() {
  // 앱 테마를 변경하는 코드
}

// html에서 이벤트 속성 이름은 모두 소문자 입니다.
<button onclick="handleToggleTheme()"> 테마 변경 </button>

// 하지만 JSX에서 이벤트 속성 이름은 camelCase로 표기합니다.
// 그리고 중괄호({})를 사용해 함수 참조를 전달 설정합니다.
<button onClick={handleToggleTheme}> 테마 변경 </button>
```

핵심 개념

🔑 이벤트 & 이벤트 핸들러 → React 이벤트 핸들러 연결 (onChange, onClick 등)

```
function App() {
  function handleChange(event) {
    // onChange와 같은 이벤트에 이벤트 핸들러(함수)를 연결할 때
    // 이벤트 데이터(객체)에 접근 할 수 있습니다.
    // event.target에서 입력한 텍스트 등을 가져올 수 있습니다.
    const inputText = event.target.value;
    const inputName = event.target.name;
  }

  return (
    <div className="form-control">
      <input name="userId" aria-label="사용자 ID" onChange={handleChange} />
      <button type="submit" onClick={handleSubmit}>전송</button>
    </div>
  );
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🛠️ 상태(state) & useState() → 함수 컴포넌트에 로컬 상태 제공

```jsx
import React from 'react';

// 상태 변수 생성
// 구문: const [상태 변수] = React.useState(기본값);
function App() {
  // 배열 변수를 사용하여 상태 변수를 선언합니다.
  const [framework] = React.useState('React');
  return <div>{framework}를 배웁니다.</div>;
}
```

```jsx
import React, { useState } from "react";

function App() {
  const [framework] = useState("React");
  return <div>{framework}를 배웁니다.</div>;
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🛠️ 상태(state) & useState() → 로컬 상태를 변경 하려면 setter 함수 사용

```
function App() {
  // setter 함수는 항상 2번째로 구조화 된 값입니다.
  // setter 함수 이름 규칙은 'setStateVariable'입니다.
  const [framework, setFramework] = React.useState("React");
  return (
    <div>
      {/*setter 함수가 호출 될 때마다 상태가 업데이트 됩니다.*/}
      {/*앱 컴포넌트가 다시 렌더링 되어 새로운 상태를 표시합니다.*/}
      <button onClick={() ⇒ setFramework("Vue")}>
        사용 할 프레임워크를 Vue로 변경
      </button>
      <p>현재 배우고 있는 프레임워크는 {framework}입니다.</p>
    </div>
  );
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🛠️ 상태(state) & useState() → 하나 이상 로컬 상태 / setter 함수 활용

```
const [framework, setFramework] = React.useState("React");
const [yearsExperience, setYearsExperience] = React.useState(0);
return (
  <div>
    <button onClick={() ⇒ setFramework("Vue")}>
      사용 할 프레임워크를 Vue로 변경
    </button>
    <input
      type="number"
      value={yearsExperience}
      onChange={event ⇒ setYearsExperience(event.target.value)}
    />
    <p>현재 사용 중인 프레임워크는 {framework}입니다.</p>
    <p>{yearsExperience} 년간 사용 했습니다.</p>
  </div>
);
```

WITH
YAMOO9
DERESA KIM

React 훅

🛠️ 상태(state) & useState() → 객체를 활용한 단 하나의 로컬 상태 관리

```
// 원시 데이터 타입 외에도 객체 데이터 타입을 활용할 수 있습니다.
const [developer, setDeveloper] = React.useState({
  framework: "",
  yearsExperience: 0
});

function handleChangeYearsExperie
  const years = event.target.valu
  // 전개 연산자를 사용하여 이전 상태 객체를
  setDeveloper({ ...developer, ye
}
```

```
<button
  onClick={() =>
    setDeveloper({
      framework: "Vue",
      yearsExperience: 0
    })
  }
></button>
```

```
<input
  type="number"
  value={developer.yearsExperience}
  onChange={handleChangeYearsExperience}
/>
```

WITH
YAMOO9
DERESA KIM

FED

React 훅

🛠️ 상태(state) & useState() → 이전 상태(prevState)를 활용한 토글

```
function App() {
  const [developer, setDeveloper] = React.useState({
    framework: "",
    yearsExperience: 0,
    isEmployed: false
  });

  function handleToggleEmployment(event) {
    setDeveloper(prevState ⇒ ({  ...prevState, isEmployed: !prevState.isEmployed }));
  }

  return (
    <button onClick={handleToggleEmployment}>고용 상태 전환</button>
  );
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🧨 사이드 이펙트(Side Effects) & useEffect → 라이프 사이클 훅을 대체하는 사이드 이펙트

```
function App() {
  const [colorIndex, setColorIndex] = React.useState(0);
  const colors = ['blue', 'green', 'red', 'orange'];

  React.useEffect(() => {
    document.body.style.backgroundColor = colors[colorIndex];
  });

  function handleChangeIndex() {
    const next = colorIndex > colors.length - 1 ? 0 : colorIndex + 1;
    setColorIndex(next);
  }

  return <button onClick={handleChangeIndex}>배경색 변경하기</button>;
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🧨 사이드 이펙트 & useEffect → 조건부 사이드 이펙트 처리 (의존성 배열 설정)

```
function App() {
  // 버튼을 몇 번 클릭 하더라도 버튼이 작동하지 않습니다.
  // 의존성 배열에서 확인할 상태가 없기 때문에 1회만 실행됩니다.
  useEffect(
    () => {
      document.body.style.backgroundColor = colors[colorIndex];
    },
    // useEffect는 의존성 배열로 조건부 콜백 함수를 실행합니다.
    []
  );
  return <button onClick={handleChangeIndex}>배경색 변경하기</button>;
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🧨 사이드 이펙트 & useEffect → 조건부 사이드 이펙트 처리 (상태 업데이트 시 콜백 함수 실행)

```
function App() {
  const [colorIndex, setColorIndex] = React.useState(0);
  const colors = ['blue', 'green', 'red', 'orange'];

  useEffect(
    () ⇒ document.body.style.backgroundColor = colors[colorIndex],
    // 의존성 배열에 colorIndex를 추가하면,
    // colorIndex가 업데이트 되었을 때 useEffect의 콜백 함수가 재 실행됩니다.
    [colorIndex]
  );
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🧨 사이드 이펙트 & useEffect → 구독 취소 (함수 반환), 컴포넌트 제거 시 실행

```
function MouseTracker() {
  const [mousePosition, setMousePosition] = useState({ x: 0, y: 0 });

  function handleMouseTrack({ pageX, pageY }) {
    setMousePosition({ x: pageX, y: pageY });
  }

  React.useEffect(() => {
    window.addEventListener("mousemove", handleMouseTrack); // 구독
    return () => {
      window.removeEventListener("mousemove", handleMouseTrack); // 구독 취소
    };
  }, []);
}
```

React 훅

🧨 사이드 이펙트 & useEffect → 프로미스를 활용한 데이터 패치 (비동기 처리)

```
const endpoint = "https://api.github.com/users/yamoo9";

function App() {
  const [user, setUser] = React.useState(null);

  React.useEffect(() ⇒ {
    // ES6 프로미스(Promise)를 활용한 콜백
    fetch(endpoint)
      .then(response ⇒ response.json())
      .then(data ⇒ setUser(data));
  }, []);
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🧨 사이드 이펙트 & useEffect → Async/Await 활용한 데이터 패치 (비동기 처리)

```
function App() {
  const [user, setUser] = React.useState(null);

  React.useEffect(() ⇒ {
    getUser();
  }, []);

  async function getUser() {
    const data = await (await fetch(endpoint)).json();
    setUser(data);
  }
}
```

WITH
YAMOO9
DERESA KIM

React 훅

🚀 성능(Performance) & useCallback → 성능 저하 방지 목적으로 사용하는 useCallback()

```
const [time, setTime] = React.useState();
const [count, setCount] = React.useState(0);

// useCallback으로 감싸지 않은 함수는 매번 다시 렌더링 할 때마다 다시 생성 (성능 저하)
// useCallback 훅은 매번 다시 생성되지 않는 콜백을 반환합니다.
const handleIncreamentCount = React.useCallback(
  () => setCount(prevCount => prevCount + 1),
    // 의존성 배열에 설정된 상태 또는 함수가 변경된 경우에만 재 실행됩니다.
  [setCount]
);

React.useEffect(() => {
  const currentTime = JSON.stringify(new Date(Date.now()));
  const timeout = setTimeout(() => setTime(currentTime), 300);
  return () => clearTimeout(timeout);
}, [time]);
```

WITH
YAMOO9
DERESA KIM

React 훅

📦 메모이제이션(Memoization) & useMemo → 성능 저하 방지 목적으로 사용하는 useMemo()

```
// useMemo는 많은 컴퓨팅 리소스가 필요할 때 유용합니다.
// 작업을 수행하지만 다시 렌더링 할 때마다 작업을 반복하지 않습니다.
const [wordIndex, setWordIndex] = useState(0);
const [count, setCount] = useState(0);
const words = ['현재', '학습 중인', '프레임워크는', 'react', '입니다.'];
const word = words[wordIndex];

function getLetterCount(word) {
  // 매우 긴(불 필요한) 루프를 사용하여 비싼 계산 시뮬레이션
  let i = 0;
  while (i < 1000000) { i++; }
  return word.length;
}

// 카운터 업데이트가 지연됩니다. 비싼 함수가 끝날 때까지 기다려야 합니다.
const letterCount = getLetterCount(word);
```

React 훅

📦 메모이제이션 & useMemo → 성능 저하 방지 목적으로 사용하는 useMemo()

```
// 입력이 같으면 비싼 값을 기억하여 이전 값을 반환
// 캐시 된 값이 없는 새로운 단어인 경우에만 계산을 수행합니다.
const letterCount = React.useMemo(() ⇒ getLetterCount(word), [word]);
```

WITH
YAMOO9
DERESA KIM

React 훅

📮 참조(Refs) & useRef → 마운트 이후, DOM / React 요소를 참조하기 위한 특수 속성

```
function App() {
  const [query, setQuery] = React.useState('React 훅');
  // useRef에 기본값을 전달할 수 있습니다.
  const searchInput = React.useRef(null);

  function handleClearSearch() {
    // 앱이 마운트 되면 현재 텍스트 입력을 참조합니다.
    // useRef는 기본적으로 .current 속성에 모든 값을 저장할 수 있습니다.
    searchInput.current.value = '';
    searchInput.current.focus();
  }

  return (
    <input type="text" ref={searchInput} onChange={event => setQuery(event.target.value)} />
  );
}
```

WITH
YAMOO9
DERESA KIM

훅 고급 활용

💠 컨텍스트(Context) & useContext → 중첩된 컴포넌트 사이 props를 여러 단계에 걸쳐 전달하는 문제

```
function App() {
  // user 데이터를 Header로 전달하고 싶습니다.
  const [user] = React.useState({ name: "야무" });

  return <Main user={user} />;
}

const Main = ({ user }) => (
  <>
    <Header user={user} />
    <div>메인 콘텐츠</div>
  </>
);

const Header = ({ user }) => <header>환영합니다. {user.name}!</header>;
```

WITH
YAMOO9
DERESA KIM

훅 고급 활용

💠 컨텍스트 & useContext → React.createContext()를 사용해 문제 해결

```
const UserContext = React.createContext();

function App() {
  return (
    <UserContext.Provider value={user}>
      <Main />
    </UserContext.Provider>
  );
}

const Header = () => (
  {/* 데이터에 접근하기 위해 render props 패턴을 활용합니다. */}
  <UserContext.Consumer>
    {user => <header>환영합니다. {user.name}!</header>}
  </UserContext.Consumer>
);
```

WITH
YAMOO9
DERESA KIM

FE
D

훅 고급 활용

💠 컨텍스트 & useContext → React.Consumer 대신 useContext() 훅 활용

```
const Header = () => {
  const user = React.useContext(UserContext);
  // Consumer 요소 및 render props를 제거 할 수 있습니다.
  return <header>환영합니다. {user.name}!</header>;
};
```

WITH
YAMOO9
DERESA KIM

훅 고급 활용

🔰 리듀서(Reducers) & useReducer → 상태 관리 라이브러리 Redux의 리듀서 패턴

```
const initialState = { username: '', isAuth: false };

function reducer(state, action) {
  switch (action.type) {
    case 'LOGIN':
      return { username: action.payload.username, isAuth: true };
    case 'SIGNOUT':
      return { username: '', isAuth: false };
    default:
      return state;
  }
}
```

WITH
YAMOO9
DERESA KIM

훅 고급 활용

🔰 리듀서 & useReducer → 상태 관리 리듀서 패턴 활용을 위한 useReducer()

```
const [state, dispatch] = useReducer(reducer, initialState);

function handleLogin() {
  dispatch({ type: "LOGIN", payload: { username: "야무" } });
}

function handleLogout() {
  dispatch({ type: "LOGOUT" });
}

return (
  <>
    현재 사용자: {state.username} / 로그인 상태: {state.isAuth}
    <button onClick={handleLogin}>로그인</button>
    <button onClick={handleLogout}>로그아웃</button>
  </>
);
```

WITH
YAMOO9
DERESA KIM

훅 고급 활용

🎮 사용자 정의 훅(Custom Hooks) → 반복되고 번거로운 절차를 처리하는 훅 생성

```
// 비동기 통신을 통해 API 데이터를 가져와 페치(Fetch)하는 사용자 정의 훅입니다.
function useAPI(endpoint) {
  const [value, setValue] = React.useState([]);

  React.useEffect(() ⇒ getData(), []);

  // 비동기 함수 getData
  async function getData() {
    const data = await (await fetch(endpoint)).json();
    // 비동기 통신이 완료 되면, 상태 업데이트
    setValue(data);
  };

  // 변경된 상태 값 반환
  return value;
};
```

WITH
YAMOO9
DERESA KIM

훅 고급 활용

🎮 사용자 정의 훅(Custom Hooks) → 정의한 훅 useAPI 사용

```
function App() {
  // 사용자 정의 훅 useAPI 사용
  // endpoint 매개변수 값으로 API URL 전달
  const users = useAPI('https://jsonplaceholder.typicode.com/users');

  return (
    <ul>
      {
        users.map(user => <li key={user.id}>{user.username}</li>
      }
    </ul>
  );
}
```

WITH
YAMOO9
DERESA KIM

훅 고급 활용

⚠️ 훅(Hooks) 작성 규칙

⚡ 훅은 조건문, 반복문, 중첩 함수 안에서는 사용할 수 없습니다.

⚡ 훅은 함수 컴포넌트 안에서만 사용할 수 있습니다. (일반 JS 함수, 클래스 컴포넌트 ❌)

WITH
YAMOO9
DERESA KIM

```
function checkAuth() {
  // 규칙2 위반!
  // 훅은 일반 JS 함수에서 사용할 수 없으며
  // 오직 함수 컴포넌트 안에서만 사용할 수 있습니다.
  React.useEffect(() => getUser(), []);
}

function App() {
  // 함수 컴포넌트 안에서 올바르게 사용된 훅입니다.
  const [user, setUser] = React.useState(null);

  // 규칙1 위반!
  // 조건 또는 반복 문 안에서는 훅을 사용할 수 없습니다.
  if (!user) {
    React.useEffect(() => setUser({ isAuth: false }), []);
  }

  checkAuth();

  // 규칙1 위반!
  // 중첩 함수 안에서 훅을 사용할 수 없습니다.
  return <button onClick={() => React.useMemo(() => doStuff(), [])}>앱</button>;
}
```

WITH
YAMOO9
DERESA KIM