# React

## 소플의 처음 만난 React

ReactJS tutorial For Beginners

# create-react-app

리액트 프로젝트 시작하기

# npm vs npx

## npm(Node Package Manager)

- npm은 Node.js 패키지를 설치, 관리하는 패키지 매니저
- package.json을 기반으로 프로젝트의 의존성을 관리
- 전역(global) 또는 로컬(local)로 패키지를 설치 가능

## npx(Node Package Execute)

- npx는 npm 패키지를 실행하는 도구
- 패키지를 설치 없이 바로 실행할 수 있음
- 프로젝트에 로컬로 설치된 패키지를 실행할 때도 사용

| 구분 | npm | npx |
|---|---|---|
| 역할 | 패키지 관리 | 패키지 실행 |
| 패키지 설치 | 가능 | 불가능 |
| 패키지 관리 | 가능 | 불가능 |
| 패키지 실행 | 로컬 또는 전역 설치된 패키지 실행 | 설치 없이 패키지 실행, 특정 버전 실행 |
| 활용 목적 | 프로젝트에 설치된 패키지 설치 및 관리 | 일회성 패키지 실행, 특정 버전 패키지 실행 |
| 사용 예제 | npm install axios | npx create-react-app myp-app |

**npm은 패키지 관리용, npx는 패키지 실행용!**

패키지는 특정 기능을 수행하는 데 필요한 코드, 리소스 및 설정 파일 등을 모아놓은 묶음을 의미합니다. 패키지는 재사용 가능하며, 모듈화된 코드를 통해 개발 생산성을 높이고 코드 관리를 용이하게 합니다.
주로 소프트웨어 개발 및 웹 개발에서 사용되는 개념으로, 관련된 코드나 리소스를 그룹화하여 관리하는 방법을 의미합니다.

# vite

- Vite(비트)는 빠르고 효율적인 프론트엔드 개발 빌드 도구입니다.
- 기존 Webpack보다 빠른 개발 서버와 빌드 속도를 제공하며, React, Vue, Svelte 등 다양한 프레임워크를 지원합니다.
- Vite는 빠르고 간결한 최신 웹 프로젝트 개발 경험에 초점을 맞춰 탄생한 차세대 프론트엔드 빌드 도구입니다. Vue.js의 창시자인 Evan You가 만들었으며, 현재 Vue, React, Svelte 등 주요 프론트엔드 라이브러리/프레임워크 커뮤니티에서 주목받고 있습니다.

# vite vs webpack

| 구분 | Vite | Webpack |
|---|---|---|
| 개발 서버 속도 | 빠름 (ESM 기반) | 느림 (번들링 필요) |
| HMR 지원 | 즉각적 반영 | 상대적으로 느림 |
| 번들링 방식 | 필요할 때만 빌드 | 전체 파일 번들링 |
| 설정 파일 | 간단한 설정 (vite.config.js) | 복잡한 설정 (webpack.config.js) |
| 사용성 | React, Vue, Svelte 등 지원 | 대부분의 프레임워크 지원 |

ESM을 통해 모듈화된 코드를 작성하고, 번들링(여러 개의 JavaScript 파일(모듈)을 하나의 파일로 묶는 과정)을 통해 최적화된 파일을 생성함으로써, 개발자는 더 나은 사용자 경험을 제공할 수 있습니다. 이러한 기술들은 앞으로도 계속 발전할 것이며, 웹 개발의 중요한 기초가 될 것입니다.

ESM을 사용하면 모듈을 네이티브로 로드할 수 있지만, 실제 배포 환경에서는 번들링을 통해 성능을 향상시키는 것이 일반적입니다.

# 리액트 프로젝트 생성

- 리액트 프로젝트를 만들기 위해 Vite 또는 Create React App (CRA) 를 사용할 수 있다.
- Vite가 속도가 더 빠르므로, 최근에는 Vite가 더 많이 사용된다.

1. VS code 실행
2. 폴더 생성 및 열기
3. 터미널(^+~)

```
# Vite 프로젝트 생성
npm create vite@latest my-react-app1 --template react

# 프로젝트 폴더로 이동
cd my-react-app1

# 의존성 설치
npm install

# 개발 서버 실행
npm run dev
```

```
# Create React App (CRA)로 설치
npx create-react-app my-react-app2

npm install web-vitals

cd my-react-app2

npm start
```

```
# Create React App (CRA)로 설치
npx create-react-app my-react-app3 --use-npm

cd my-react-app3

npm start
```

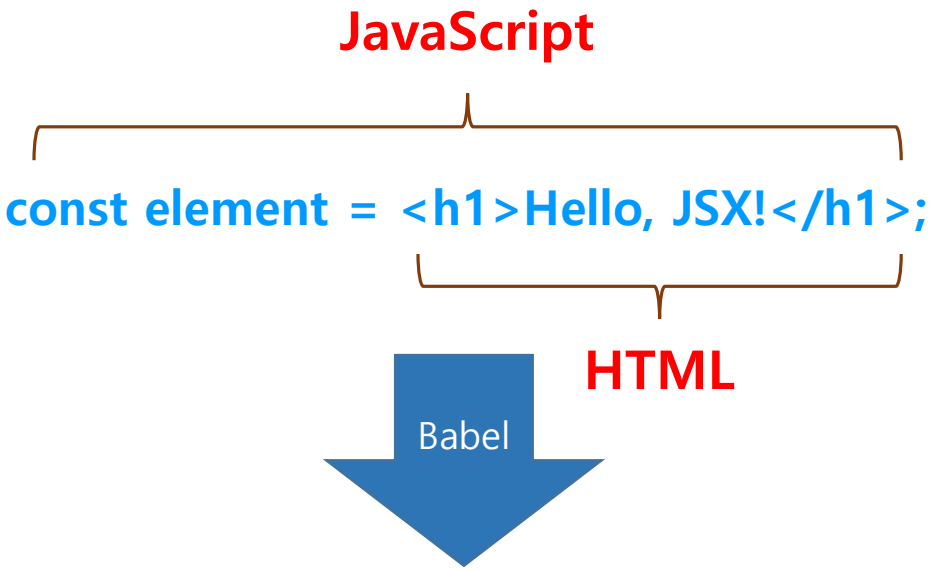
# JavaScript XML

JSX is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file

JavaScript 코드 안에서 HTML과 유사한 문법을 사용할 수 있도록 만든 문법 확장

# JSX

- HTML과 JavaScript를 결합한 문법
- Babel을 이용해 JavaScript로 변환됨
- React 요소를 생성하는 데 사용됨
- JavaScript 표현식을 {}로 감싸서 사용 가능
- 클래스 대신 className, for 대신 htmlFor 사용
- …




```
const element = React.createElement("h1", null, "Hello, JSX!");
```

✓ JSX는 React에서 UI를 선언적으로 작성할 수 있도록 도와주는 문법이며, JavaScript 코드 안에서 HTML과 유사한 태그를 사용할 수 있게 해줍니다.

✓ Babel을 사용하여 일반 JavaScript로 변환되며, 컴포넌트 기반 개발을 쉽게 할 수 있도록 도와줍니다!

✓ JSX를 사용하는 것이 필수는 아닙니다.

## JSX를 사용한 코드

```
import React from "react";
import ReactDOM from "react-dom";
function App() {
  return (
    <div>
      <h1>Hello, JSX!</h1>
      <p>This is a JSX example.</p>
    </div>
  );
}
ReactDOM.createRoot(document.getElementById("root")).render(<App />);
```

| 비교 | JSX 사용 | JSX 미사용 |
|---|---|---|
| 문법 | HTML과 유사 | JavaScript의 함수 호출 형식 |
| 가독성 | 직관적이고 간결함 | 코드가 복잡하고 가독성이 낮음 |
| 요소 생성 방식 | 태그( , <h1>) 사용 | React.createElement() 호출 |
| 유지보수 | 쉬움 | 어려움 (중첩 구조 복잡) |
| 변환 과정 | Babel이 JavaScript로 변환 | 직접 JavaScript 코드로 작성 |

## JSX를 사용하지 않은 코드

```
import React from "react";
import ReactDOM from "react-dom";

function App() {
  return React.createElement(
    "div",
    null,
    React.createElement("h1", null, "Hello, JSX!"),
    React.createElement("p", null, "This is a JSX example.")
  );
}

ReactDOM.createRoot(document.getElementById("root")).render(React.createElement(App));
```

createElement()

# JSX 장점

## 1. 가독성이 좋고 직관적임

```
function App() {
  return (
    <div>
      <h1>Hello, React!</h1>
      <p>JSX를 사용하면 가독성이 좋아집니다.</p>
    </div>
  );
}
```

```
function App() {
  return React.createElement(
    "div",
    null,
    React.createElement("h1", null, "Hello, React!"),
    React.createElement("p", null, "JSX를 사용하면 가독성이 좋아집니다.")
  );
}
```

## 2. 코드가 간결하고 유지보수가 쉬움

```
function List() {
  return (
    <ul>
      <li>React</li>
      <li>Vue</li>
      <li>Angular</li>
    </ul>
  );
}
```

```
function List() {
  return React.createElement(
    "ul",
    null,
    React.createElement("li", null, "React"),
    React.createElement("li", null, "Vue"),
    React.createElement("li", null, "Angular")
  );
}
```

# JSX 장점

## 3. 자바스크립트와 자연스럽게 결합 가능

```
const name = "React";
function Greeting() {
  return <h1>Hello, {name}!</h1>;
}
```

```
const name = "React";
function Greeting() {
  return React.createElement("h1", null, `Hello, ${name}!`);
}
```

## 4. 조건부 렌더링이 쉬움

```
function Greeting({ isLoggedIn }) {
  return <h1>{isLoggedIn ? "Welcome back!" : "Please log in"}</h1>;
}
```

```
function Greeting({ isLoggedIn }) {
  return React.createElement(
    "h1",
    null,
    isLoggedIn ? "Welcome back!" : "Please log in"
  );
}
```

# JSX 장점

## 5. 스타일 및 이벤트 핸들링이 쉬움

```
const buttonStyle = { backgroundColor: "blue", color: "white" };

function Button() {
  return <button style={buttonStyle}>Click me</button>;
}
```

```
const buttonStyle = { backgroundColor: "blue", color: "white" };

function Button() {
  return React.createElement("button", { style: buttonStyle }, "Click me");
}
```

**JSX는 React에서 UI를 정의하는 데 있어 많은 장점을 제공합니다. 가독성이 높고, JavaScript 표현식을 사용할 수 있으며, 컴포넌트를 쉽게 재사용하고 조건부 렌더링을 구현할 수 있습니다. 이러한 이유로 많은 개발자들이 JSX를 선호합니다.**

# JSX 사용법

## 1. JSX 기본 구조

JSX에서는 HTML과 유사한 문법을 사용하여 요소를 반환할 수 있다.
JSX는 반드시 하나의 부모 요소로 감싸야 한다.

```
function App() {
  return (
    <h1>Hello, JSX!</h1>
    <p>React에서 JSX를 사용합니다.</p>  // 에러 발생
  );
}
```

```
function App() {
  return (
    <div>
      <h1>Hello, JSX!</h1>
      <p>React에서 JSX를 사용합니다.</p>
    </div>
  );
}
```

```
function App() {
  return (
    <div>
      <h1>Hello, JSX!</h1>
      <p>React에서 JSX를 사용합니다.</p>
    </div>
  );
}
```

```
function App() {
  return (
    <>
      <h1>Hello, JSX!</h1>
      <p>React에서 JSX를 사용합니다.</p>
    </>
  );
}
```

# JSX 사용법

## 2. JSX에서 자바스크립트 표현식

JSX 내부에서 JavaScript 표현식을 {}로 감싸서 사용할 수 있다.

```
const name = "React";

function App() {
  return <h1>Hello, {name}!</h1>;
}
```

```
function App() {
  return <p>2 + 3 = {2 + 3}</p>;
}
```

```
function getGreeting(name) {
  return `Hello, ${name}!`;
}

function App() {
  return <h1>{getGreeting("React")}</h1>;
}
```

# JSX 사용법

## 3. JSX에서 속성(props) 사용

JSX에서는 HTML 속성과 유사한 방식으로 속성을 설정할 수 있지만, 일부 속성은 JavaScript 문법을 따른다.

문자열 속성

```
const imgUrl = "https://via.placeholder.com/150";

function App() {
  return <img src={imgUrl} alt="Sample Image" />;
}
```

클래스 설정(className 사용)

```
function App() {
  return <h1 className="title">Hello, JSX!</h1>;
}
```

객체 속성 사용(스타일 적용)

```
const styleObj = { color: "blue", fontSize: "20px" };

function App() {
  return <h1 style={styleObj}>Styled Text</h1>;
}
```

# JSX 사용법

## 4. JSX에서 조건부 렌더링

JSX에서는 삼항 연산자를 사용하여 조건부 렌더링을 구현할 수 있다.

삼항 연산자를 이용한 조건부 렌더링

```
function App({ isLoggedIn }) {
  return <h1>{isLoggedIn ? "Welcome back!" : "Please log in"}</h1>;
}
```

&& 연산자를 이용한 조건부 렌더링

```
function App({ isAdmin }) {
  return (
    <div>
      <h1>Hello, User!</h1>
      {isAdmin && <p>관리자 권한이 있습니다.</p>}
    </div>
  );
}
```

# JSX 사용법

## 5. JSX에서 반복문 사용 (배열과 map())

JSX에서는 for문을 직접 사용할 수 없고, map() 메서드를 이용해 반복 렌더링을 해야 한다.

```
const fruits = ["🍎 Apple", "🍌 Banana", "🍊 Orange"];

function App() {
  return (
    <ul>
      {fruits.map((fruit, index) => (
        <li key={index}>{fruit}</li> // key 속성 필수
      ))}
    </ul>
  );
}
```

key 속성을 반드시 추가해야 React가 효율적으로 렌더링할 수 있다.

# JSX 사용법

## 6. JSX에서 이벤트 처리

JSX에서는 이벤트를 onClick, onChange 등의 속성으로 설정할 수 있다.

```
function App() {
  function handleClick() {
    alert("버튼이 클릭되었습니다!");
  }

  return <button onClick={handleClick}>Click Me</button>;
}
```

```
function App() {
  function handleChange(event) {
    console.log("입력값:", event.target.value);
  }

  return <input type="text" onChange={handleChange} />;
}
```

- **JSX는 React에서 UI를 직관적으로 작성할 수 있도록 도와주는 JavaScript 확장 문법이다.**
- **JSX 내부에서 {}를 사용하여 JavaScript 표현식을 활용할 수 있다.**
- **JSX를 사용하면 조건부 렌더링, 반복 렌더링, 이벤트 처리 등이 편리해진다.**
- **JSX는 HTML과 유사하지만 JavaScript의 규칙을 따르므로 className, style={} 등의 차이점을 주의해야 한다.**

# JSX 코드 작성해보기

1. VS Code 실행하기
2. 터미널 열기(&~)
3. 프로젝트 생성 및 폴더 이동

    3.1 npx create-react-app react-app-ex01 --use-npm

    3.2 cd react-app-ex01

4. react-app-ex01 프로젝트의 src 폴더에 chapter_03 폴더 생성
5. chapter_03 폴더에 Book.jsx 파일 생성

Book.jsx

```
import React from "react";
function Book(props) {
    return (
        <div>
            <h1>{`이 책의 이름은 ${props.name}입니다.`}</h1>
            <h2>{`이 책은 총 ${props.numOfPage}페이지로 이뤄져 있습니다.`}</h2>
        </div>
    );
}
export default Book;
```

# JSX 코드 작성해보기

## 6. chapter_03 폴더에 Library.jsx 파일 생성

Library.jsx

```
import React from "react";
import Book from "./Book";

function Library(props) {
    return (
        <div>
            <Book name="처음 만난 파이썬" numOfPage={300} />
            <Book name="처음 만난 AWS" numOfPage={400} />
            <Book name="처음 만난 리액트" numOfPage={500} />
        </div>
    );
}

export default Library;
```

# JSX 코드 작성해보기

## 7. react-app-ex01 프로젝트의 src 폴더에 있는 index.js 편집

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

import Library from './chapter_03/Library';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <Library />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

이 책의 이름은 처음 만난 파이썬입니다.

이 책은 총 300페이지로 이뤄져 있습니다.

이 책의 이름은 처음 만난 AWS입니다.

이 책은 총 400페이지로 이뤄져 있습니다.

이 책의 이름은 처음 만난 리액트입니다.

이 책은 총 500페이지로 이뤄져 있습니다.

## 8. 터미널에서 npm start 실행