

## # React 스타일링 방법 완벽 가이드

React에서 컴포넌트를 스타일링하는 다양한 방법을 소개합니다.

### ## 목차

1. [일반 CSS](#1-일반-css)
2. [CSS Modules](#2-css-modules)
3. [SCSS/SASS](#3-scsssass)
4. [Styled-Components](#4-styled-components)
5. [Emotion](#5-emotion)
6. [Tailwind CSS](#6-tailwind-css)
7. [Inline Styles](#7-inline-styles)
8. [비교표](#비교표)

---

### ## 1. 일반 CSS

가장 전통적인 방법으로, 별도의 CSS 파일을 만들어 import하는 방식입니다.

```
### 설치
```bash
# 별도 설치 불필요 (기본 지원)
```
```

### ### 사용법

```
**Button.jsx**
```jsx
import React from 'react';
import './Button.css';

function Button({ children }) {
  return <button className="button">{children}</button>;
}

export default Button;
```
```

```
**Button.css**
```css
.button {
  padding: 10px 20px;
  background-color: #3498db;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}```
```

```
}
```

```
.button:hover {
```

```
    background-color: #2980b9;
```

```
}
```

```
..
```

### ### 장점

- ✓ 배우기 쉬움
- ✓ 별도 설치 불필요
- ✓ 익숙한 CSS 문법
- ✓ 디자이너와 협업 용이

### ### 단점

- ✗ 전역 스코프 (클래스명 충돌 가능)
- ✗ 사용하지 않는 CSS도 로드됨
- ✗ 동적 스타일링 어려움

### ### 적합한 경우

- 간단한 프로젝트
- CSS에 익숙한 팀
- 전역 스타일이 많이 필요한 경우

---

## ## 2. CSS Modules

일반 CSS의 클래스명 충돌 문제를 해결한 방식입니다.

```
### 설치
```

```
```bash
```

```
# Create React App, Vite는 기본 지원
```

```
# Next.js도 기본 지원
```

```
..
```

### ### 사용법

```
**Button.jsx**
```

```
```jsx
```

```
import React from 'react';
import styles from './Button.module.css';

function Button({ children, variant = 'primary' }) {
  return (
    <button className={`${styles.button} ${styles[variant]}`}>
      {children}
    </button>
  );
}
```

```

}

export default Button;
```
**Button.module.css**
```css
.button {
  padding: 10px 20px;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}

.primary {
  background-color: #3498db;
  color: white;
}

.secondary {
  background-color: #95a5a6;
  color: white;
}
```

### 장점
- ✓ 로컬 스코프 (클래스명 자동 변환)
- ✓ 클래스명 충돌 방지
- ✓ 익숙한 CSS 문법
- ✓ 타입스크립트 지원 가능

### 단점
- ✗ 클래스명이 복잡해짐 (디버깅 시)
- ✗ 동적 스타일링 번거로움
- ✗ 전역 스타일 사용 시 :global() 필요

### 적합한 경우
- 중소 규모 프로젝트
- 컴포넌트별 스타일 격리가 필요한 경우
- CSS 문법을 유지하면서 모듈화가 필요한 경우
---
```

## ## 3. SCSS/SASS

CSS의 확장 버전으로, 변수, 중첩, 믹스인 등 강력한 기능을 제공합니다.

### ### 설치

```
```bash
npm install sass
# 또는
npm install -D sass
```

### 사용법

**Button.jsx**
```jsx
import React from 'react';
import './Button.scss';

function Button({ children, size = 'medium' }) {
    return <button className={`btn btn--${size}`}>{children}</button>;
}

export default Button;
```

**Button.scss**
```scss
// 변수
$primary-color: #3498db;
$button-padding: 10px 20px;

// 믹스인
@mixin button-size($padding) {
    padding: $padding;
}

// 스타일
.btn {
    background-color: $primary-color;
    color: white;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: all 0.3s ease;

    &:hover {
        background-color: darken($primary-color, 10%);
    }
}

// BEM 방식
&--small {
    @include button-size(6px 12px);
    font-size: 12px;
}
```

```

```
}

&--medium {
  @include button-size(10px 20px);
  font-size: 14px;
}

&--large {
  @include button-size(14px 28px);
  font-size: 16px;
}
}

```

```

### ### SCSS Modules 사용

#### \*\*Button.module.scss\*\*

```
```scss
.button {
  padding: 10px 20px;

  &:hover {
    opacity: 0.8;
  }
}

```

```

### ### 장점

- ✓ 변수, 믹스인, 중첩 등 강력한 기능
- ✓ 코드 재사용성 향상
- ✓ CSS Modules와 함께 사용 가능
- ✓ 대규모 프로젝트에 적합

### ### 단점

- ✗ 컴파일 필요
- ✗ 학습 곡선 존재
- ✗ 빌드 시간 증가 가능

### ### 적합한 경우

- 중대형 프로젝트
- 복잡한 스타일 로직이 필요한 경우
- 디자인 시스템 구축 시

---

## ## 4. Styled-Components

CSS-in-JS 방식의 가장 인기 있는 라이브러리입니다.

```
### 설치
```bash
npm install styled-components
```

```

```
### 사용법
```

```
```jsx
import React from 'react';
import styled from 'styled-components';

// 스타일 정의
const StyledButton = styled.button`
  padding: 10px 20px;
  background-color: ${props => props.primary ? '#3498db' : '#95a5a6'};
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: ${props => props.size === 'large' ? '16px' : '14px'};

  &:hover {
    opacity: 0.8;
  }

  &:disabled {
    opacity: 0.5;
    cursor: not-allowed;
  }
`;

// 컴포넌트
function Button({ children, primary, size }) {
  return (
    <StyledButton primary={primary} size={size}>
      {children}
    </StyledButton>
  );
}

export default Button;
```

```

```
### 고급 예제
```

```
```jsx
import styled, { css } from 'styled-components';

// props에 따른 동적 스타일

```

```
const Button = styled.button`  
  padding: 10px 20px;  
  border: none;  
  border-radius: 4px;  
  cursor: pointer;  
  
  ${props => props.variant === 'primary' && css`  
    background-color: #3498db;  
    color: white;  
  `}  
  
  ${props => props.variant === 'secondary' && css`  
    background-color: #95a5a6;  
    color: white;  
  `}  
  
  ${props => props.fullWidth && css`  
    width: 100%;  
  `}  
`;  
  
// 스타일 상속  
const PrimaryButton = styled(Button)`  
  background-color: #3498db;  
  color: white;  
`;  
  
// 테마 사용  
const theme = {  
  colors: {  
    primary: '#3498db',  
    secondary: '#2ecc71'  
  }  
};  
  
const ThemedButton = styled.button`  
  background-color: ${props => props.theme.colors.primary};  
  color: white;  
`;  
  
// 사용  
import { ThemeProvider } from 'styled-components';  
  
<ThemeProvider theme={theme}>  
  <ThemedButton>Themed Button</ThemedButton>  
</ThemeProvider>  
```
```

### ### 장점

- ✓ 컴포넌트와 스타일이 같은 파일에
- ✓ 동적 스타일링 쉬움
- ✓ Props 기반 스타일링
- ✓ 자동 벤더 프리픽스
- ✓ 사용하지 않는 스타일 제거
- ✓ 테마 기능 내장

### ### 단점

- ✗ 번들 크기 증가 (~15KB)
- ✗ 런타임 오버헤드
- ✗ 디버깅 어려움 (클래스명 자동 생성)
- ✗ SSR 설정 복잡

### ### 적합한 경우

- 동적 스타일링이 많이 필요한 경우
- 컴포넌트 중심 개발
- 테마 기능이 필요한 경우

---

## ## 5. Emotion

Styled-Components와 유사한 CSS-in-JS 라이브러리로, 더 빠르고 가볍습니다.

### ### 설치

```
```bash
npm install @emotion/react @emotion/styled
```
```

### ### 사용법 1: styled 사용

```
```jsx
import styled from '@emotion/styled';

const Button = styled.button`
  padding: 10px 20px;
  background-color: ${props => props.primary ? '#3498db' : '#95a5a6'};
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;

  &:hover {
    opacity: 0.8;
  }
`;
```

```
function App() {
  return <Button primary>Click Me</Button>;
}
```

```

### 사용법 2: css prop 사용

```
```jsx
/** @jsxImportSource @emotion/react */
import { css } from '@emotion/react';

const buttonStyle = css`  

  padding: 10px 20px;  

  background-color: #3498db;  

  color: white;  

  border: none;  

  border-radius: 4px;  

  cursor: pointer;  

  

  &:hover {  

    opacity: 0.8;  

  }
`;

function Button({ children }) {
  return <button css={buttonStyle}>{children}</button>;
}
```

```

### 동적 스타일

```
```jsx
/** @jsxImportSource @emotion/react */
import { css } from '@emotion/react';

function Button({ children, primary }) {
  return (
    <button
      css={css`  

        padding: 10px 20px;  

        background-color: ${primary ? '#3498db' : '#95a5a6'};  

        color: white;  

        border: none;  

        border-radius: 4px;  

  

        &:hover {  

          opacity: 0.8;  

        }
      `}
    >{children}</button>
  );
}
```

```
>
  {children}
</button>
);
}
```

```

### ### 장점

- ✓ Styled-Components보다 빠름
- ✓ 더 작은 번들 크기
- ✓ css prop 지원 (더 직관적)
- ✓ SSR 지원 우수
- ✓ TypeScript 지원 좋음

### ### 단점

- ✗ Styled-Components보다 커뮤니티 작음
- ✗ 학습 곡선 존재

### ### 적합한 경우

- 성능이 중요한 프로젝트
- Styled-Components의 가벼운 대안이 필요한 경우

---

## ## 6. Tailwind CSS

유틸리티 우선(Utility-First) CSS 프레임워크입니다.

### ### 설치

```
```bash
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

```

### ### 설정

```
**tailwind.config.js**
```js
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        primary: '#3498db',
      },
    }
  }
}
```

```

```

        },
    },
    plugins: [],
}
```
**index.css**
```css
@tailwind base;
@tailwind components;
@tailwind utilities;
```

### 사용법

```jsx
import React from 'react';

function Button({ children, variant = 'primary', size = 'medium' }) {
  const baseClasses = "font-semibold rounded transition-all";

  const variantClasses = {
    primary: "bg-blue-500 hover:bg-blue-600 text-white",
    secondary: "bg-gray-500 hover:bg-gray-600 text-white",
    danger: "bg-red-500 hover:bg-red-600 text-white",
  };

  const sizeClasses = {
    small: "px-3 py-1.5 text-sm",
    medium: "px-5 py-2.5 text-base",
    large: "px-7 py-3.5 text-lg",
  };

  return (
    <button
      className={`${baseClasses} ${variantClasses[variant]} ${sizeClasses[size]}`}
      >
      {children}
    </button>
  );
}

export default Button;
```

```

### 복잡한 레이아웃 예제

```jsx

```

function Card({ title, description, image }) {
  return (
    <div className="max-w-sm rounded-lg overflow-hidden shadow-lg
    hover:shadow-xl transition-shadow">
      <img className="w-full h-48 object-cover" src={image} alt={title} />
      <div className="px-6 py-4">
        <h3 className="font-bold text-xl mb-2 text-gray-900">{title}</h3>
        <p className="text-gray-700 text-base">{description}</p>
      </div>
      <div className="px-6 pt-4 pb-6">
        <button className="bg-blue-500 hover:bg-blue-700 text-white font-
        bold py-2 px-4 rounded">
          더 보기
        </button>
      </div>
    </div>
  );
}
```

```

### ### 장점

- ✓ 빠른 개발 속도
- ✓ 일관된 디자인
- ✓ 작은 최종 CSS 파일 (PurgeCSS)
- ✓ 반응형 디자인 쉬움
- ✓ 커스터마이징 용이

### ### 단점

- ✗ HTML이 복잡해짐 (클래스 많음)
- ✗ 학습 곡선 (유ти리티 클래스 암기)
- ✗ 커스텀 디자인 시 설정 필요
- ✗ 초기 설정 복잡

### ### 적합한 경우

- 빠른 프로토타이핑
- 일관된 디자인 시스템이 필요한 경우
- 반응형 웹 개발

---

## ## 7. Inline Styles

JavaScript 객체로 스타일을 직접 정의하는 방식입니다.

### ### 사용법

```

```jsx
import React from 'react';

```

```
function Button({ children, primary }) {
  const buttonStyle = {
    padding: '10px 20px',
    backgroundColor: primary ? '#3498db' : '#95a5a6',
    color: 'white',
    border: 'none',
    borderRadius: '4px',
    cursor: 'pointer',
    fontSize: '14px',
  };

  return <button style={buttonStyle}>{children}</button>;
}

export default Button;
```

```

```
### 동적 스타일
```jsx
function Button({ children, primary, disabled }) {
  return (
    <button
      style={{
        padding: '10px 20px',
        backgroundColor: disabled ? 'ccc' : primary ? '#3498db' : '#95a5a6',
        color: 'white',
        border: 'none',
        borderRadius: '4px',
        cursor: disabled ? 'not-allowed' : 'pointer',
        opacity: disabled ? 0.6 : 1,
      }}
      disabled={disabled}
    >
      {children}
    </button>
  );
}
```

```

```
### 장점
- ✓ 설치 불필요
- ✓ 동적 스타일링 쉬움
- ✓ 로컬 스코프
- ✓ JavaScript 변수 사용 가능
```

```
### 단점
- ✗ 의사 클래스 불가 (:hover, :active 등)
```

- ✕ 미디어 쿼리 불가
- ✕ 자동완성 제한적
- ✕ 성능 이슈 (매 렌더링마다 객체 생성)
- ✕ CSS의 모든 기능 사용 불가

#### ### 적합한 경우

- 간단한 동적 스타일
- 프로토타이핑
- 매우 작은 컴포넌트

---

#### ## 비교표

| 방법                    | 설치     | 학습 난이도   | 번들 크기  | 성능   | 동적 스타일 | 타입 지원       | 인기도           |
|-----------------------|--------|----------|--------|------|--------|-------------|---------------|
| **일반 CSS**            | ✕ 불필요  | ☆ 쉬움     | ✓ 작음   | ✓ 빠름 | ✕ 어려움  |             |               |
| ✕ 없음                  | ☆☆☆☆☆  |          |        |      |        |             |               |
| **CSS Modules**       | ✕ 불필요* | ☆☆ 쉬움    | ✓ 작음   | ✓ 빠름 | ⚠ 보통   | ⚠ 제한적       | ☆☆☆☆          |
| **SCSS/SASS**         | ✓ 필요   | ☆☆ 보통    | ✓ 작음   | ✓ 빠름 | ⚠ 보통   | ✕ 없음        | ☆☆☆☆☆         |
| **Styled-Components** | ✓ 필요   | ☆☆☆ 보통   | ⚠ 중간   | ⚠ 보통 |        | ✓ 쉬움        | ✓ 좋음   ☆☆☆☆☆  |
| **Emotion**           | ✓ 필요   | ☆☆☆ 보통   | ✓ 작음   | ✓ 빠름 | ✓ 쉬움   | ✓ 좋음   ☆☆☆☆ |               |
| **Tailwind CSS**      | ✓ 필요   | ☆☆☆☆ 어려움 | ✓ 작음** | ✓ 빠름 |        | ✓ 쉬움        | ⚠ 제한적   ☆☆☆☆☆ |
| **Inline Styles**     | ✕ 불필요  | ☆ 쉬움     | ✓ 작음   | ⚠ 보통 | ✓ 쉬움   | ✓ 좋음   ☆☆   |               |

W\* CRA, Vite, Next.js는 기본 지원

W\*W\* PurgeCSS 사용 시

---

#### ## 선택 가이드

##### ### 프로젝트 규모별

###### #### 소규모 (개인 프로젝트, 프로토타입)

- \*\*추천\*\*: 일반 CSS, Inline Styles, Tailwind CSS
- \*\*이유\*\*: 빠른 개발, 간단한 설정

###### #### 중규모 (스타트업, 중소기업)

- \*\*추천\*\*: CSS Modules, SCSS, Styled-Components
- \*\*이유\*\*: 유지보수성과 확장성의 균형

#### #### 대규모 (대기업, 복잡한 애플리케이션)

- \*\*추천\*\*: SCSS + CSS Modules, Styled-Components, Emotion
- \*\*이유\*\*: 디자인 시스템, 팀 협업, 확장성

#### ### 팀 구성별

##### #### CSS에 익숙한 팀

- \*\*추천\*\*: CSS Modules, SCSS

##### #### React에 익숙한 개발자

- \*\*추천\*\*: Styled-Components, Emotion

##### #### 풀스택 개발자

- \*\*추천\*\*: Tailwind CSS

#### ### 요구사항별

##### #### 빠른 개발 속도가 중요한 경우

- \*\*추천\*\*: Tailwind CSS

##### #### 성능이 최우선인 경우

- \*\*추천\*\*: CSS Modules, SCSS

##### #### 동적 스타일링이 많은 경우

- \*\*추천\*\*: Styled-Components, Emotion

##### #### 디자인 시스템 구축

- \*\*추천\*\*: SCSS, Styled-Components + Theme

----

#### ## 실전 조합 추천

##### ### 조합 1: CSS Modules + SCSS

```
```bash
npm install -D sass
```
```

- 모듈화 + 강력한 CSS 기능
- 성능 우수
- 학습 곡선 낮음

##### ### 조합 2: Styled-Components + Tailwind

```
```bash
npm install styled-components tailwindcss
```
```

- 컴포넌트 스타일링 + 유ти리티
- 빠른 개발
- 유연성 높음

### ### 조합 3: Emotion + Tailwind

```
```bash
npm install @emotion/react @emotion/styled tailwindcss
```
```

- 성능 + 빠른 개발
- 현대적인 스택

----

## ## 마무리

각 방법은 장단점이 있으며, **\*\*정답은 없습니다\*\***. 프로젝트의 요구사항, 팀의 경험, 개발 기간 등을 고려하여 선택하세요.

### ### 개인적 추천

- **입문자**: CSS Modules
- **중급자**: SCSS + CSS Modules
- **고급자**: Styled-Components 또는 Emotion
- **빠른 개발**: Tailwind CSS

행복한 코딩 되세요! 🎨✨