React

◆ 컴포넌트 스타일링-1

정수아

Contents

01 컴포넌트 스타일링

컴포넌트 스타일링

❖ 리액트 컴포넌트 스타일링

스타일링 방식	설명
일반 CSS	• 컴포넌트를 스타일링하는 가장 기본적인 방식
Sass	• 자주 사용되는 CSS 전처리기(pre-processor) 중 하나 • 확장된 CSS 문법을 사용하여 CSS 코드를 더욱 쉽게 작성할 수 있음
CSS Module	• CSS 클래스가 다른 CSS 클래스의 이름과 절대 충돌하지 않도록 파일마다 고유한 이름을 자동으로 생성해주는 옵션
styled-components	• 스타일을 자바스크립트 파일에 내장시키는 방식 • 스타일을 작성함과 동시에 해당 스타일이 적용된 컴포넌트 생성 가능

[실습] 클래스 이름 작성의 중요성

❖ Button1.js

```
import "./Button1.css";

const Button1 = () => {
  return <button className="button">버튼1</button>;
};

export default Button1;
```

❖ Button1.css

```
.button {
  font-size: 30px;
  color: white;
  background: blue;
}
```

[실습] 클래스 이름 작성의 중요성

❖ Button2.js

```
import "./Button2.css";

const Button2 = () => {
  return <button className="button">버튼2</button>;
};

export default Button2;
```

❖ Button2.css

```
.button {
  font-size: 30px;
  color: white;
  background: palevioletred;
}
```

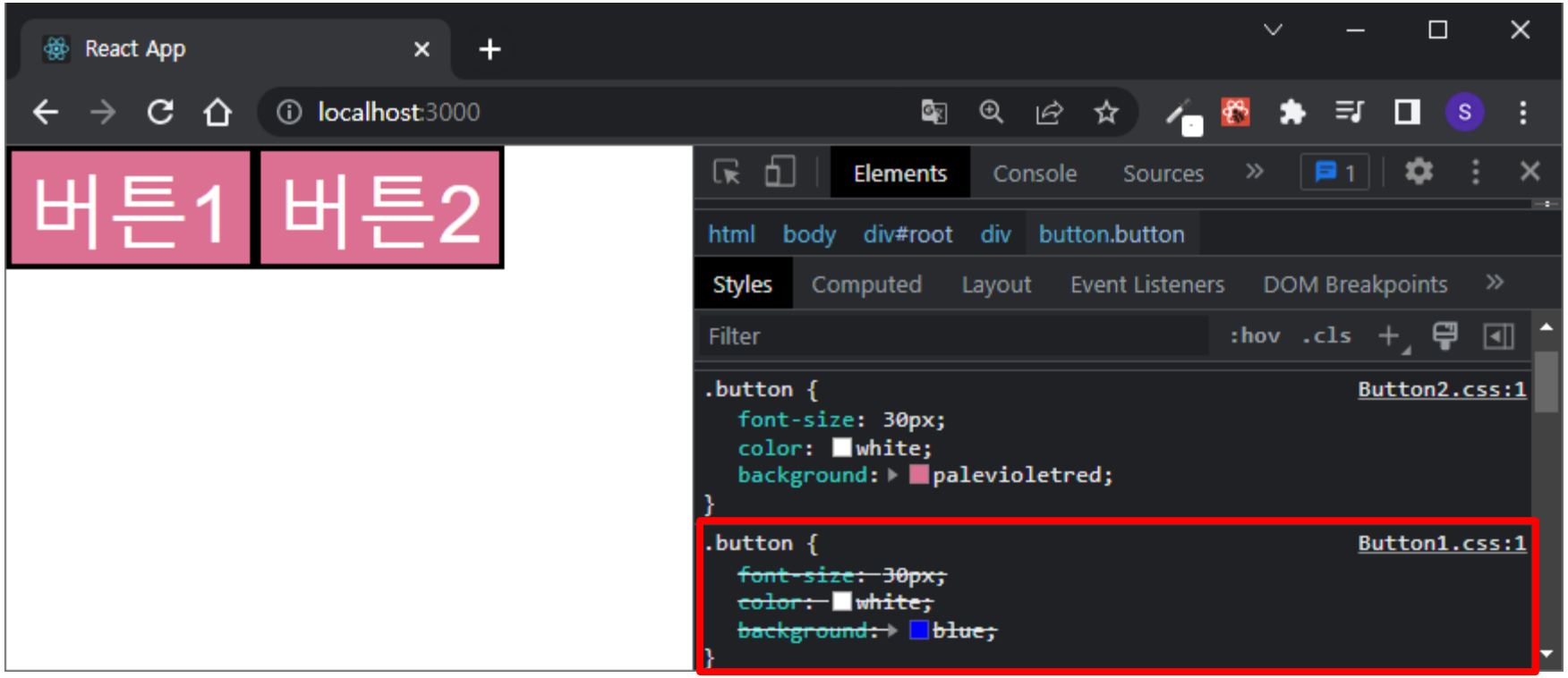
[실습] 클래스 이름 작성의 중요성

❖ **App.js**

```
import Button1 from "./components/Button1";
import Button2 from "./components/Button2";

function App() {
  return (
    <div>
      <Button1 />
      <Button2 />
    </div>
  );
}

export default App;
```

[실습] 클래스 이름 작성의 중요성

❖ 실행 결과



이름 생성 규칙

❖ **컴포넌트이름-클래스 형태**

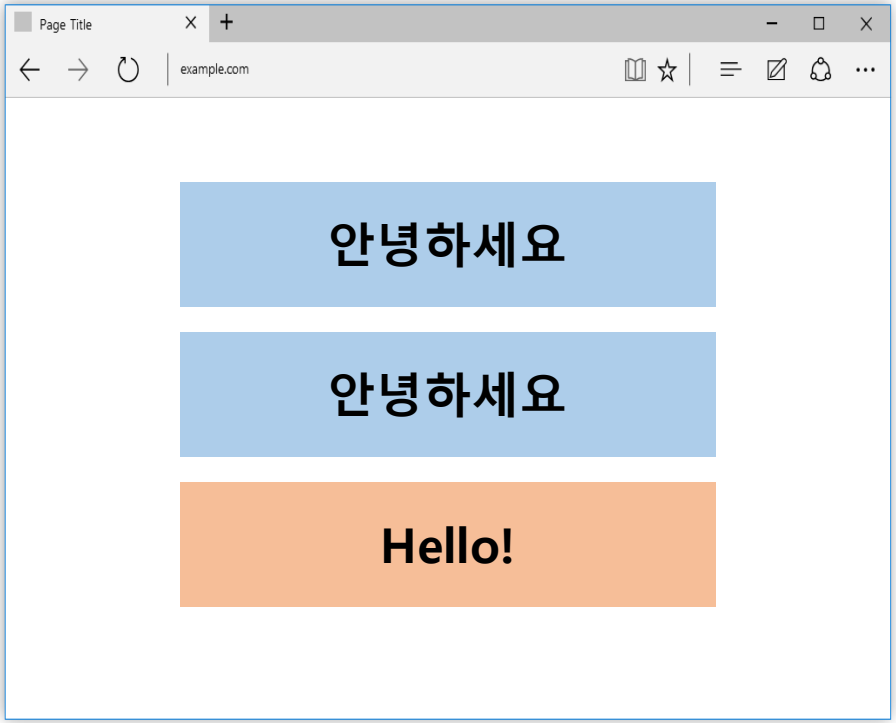
- 프로젝트 내 App.css

```
.App-logo {
   height: 40vmin;
   pointer-events: none;
}

.App-link {
   color: #61dafb;
}
```

이름 생성 규칙

❖ BEM 네이밍 방식

- Block, Element, Modifier




```
<ul class="nav">
  <li class="nav__list-item">
     안녕하세요
  </li>
  <li class="nav__list-item">
     안녕하세요
  </li>
  <li class="nav__list-item
             nav__list-item--special">
     Hello!
  </li>
</ul>
```

이름 생성 규칙

❖ BEM 네이밍 방식

```
<ul class="nav">

  <li class="nav__list-item">
     안녕하세요
  </li>

  <li class="nav__list-item">
     안녕하세요
  </li>

  <li class="nav__list-item nav__list-item--special">
     Hello!
  </li>

</ul>
```

Block

Element

Modifier

이름 생성 규칙

❖ BEM 네이밍 작성 규칙

```
Block__Element--Modifier
```

```
nav__list-item--special
```

02

Sass란?

Sass(Syntactically Awesome Style Sheets)

❖ **CSS의 단점**

- 선택자(Selector)를 만들 때 매번 부모 요소 선택자를 넣어줘야 함
- 함수 기능 제공 안함
  - 중복되는 스타일 코드가 많음
  - 코드 양 증가로 유지보수가 어려움

Sass(Syntactically Awesome Style Sheets)

❖ **Sass란?**

- 문법적으로 매우 멋진 스타일시트를 의미

- 특징
  - CSS 전처리기로 복잡한 작업을 쉽게 할 수 있음
  - 스타일 코드의 재활용성을 높임
  - 스타일 코드의 가독성을 높여서 유지 보수가 쉬움

- 확장자
  - .scss
  - .sass

CSS 전처리기(Preprocessor)

확장자

❖ .sass

```
$font-stack : Helvetica, sans-serif
$primary-color : #333

body
    font : 100% $font-stack
    color : $primary-color
```

❖ .scss

```
$font-stack : Helvetica, sans-serif;
$primary-color : #333;

body {
    font : 100% $font-stack;
    color : $primary-color;
}
```

믹스인(mixin) 문법

❖ **믹스인(mixin)**

- 재사용 가능한 기능을 만드는 방식

구분	선언	적용
.sass	=	+
.scss	@mixin	@include

믹스인(mixin) 문법

.sass

```
$main-font : "Helvetica"

=title($font)
    font-size : 30px
    font-family : $font

#header
    +title($main-font)
```

.scss

```
$main-font: "Helvetica";

@mixin title($font) {
  font-size: 30px;
  font-family: $font;
}

#header {
  @include title($main-font);
}
```

Sass 설치

❖ **node-sass 라이브러리**

- Sass를 CSS로 컴파일해주는 모듈

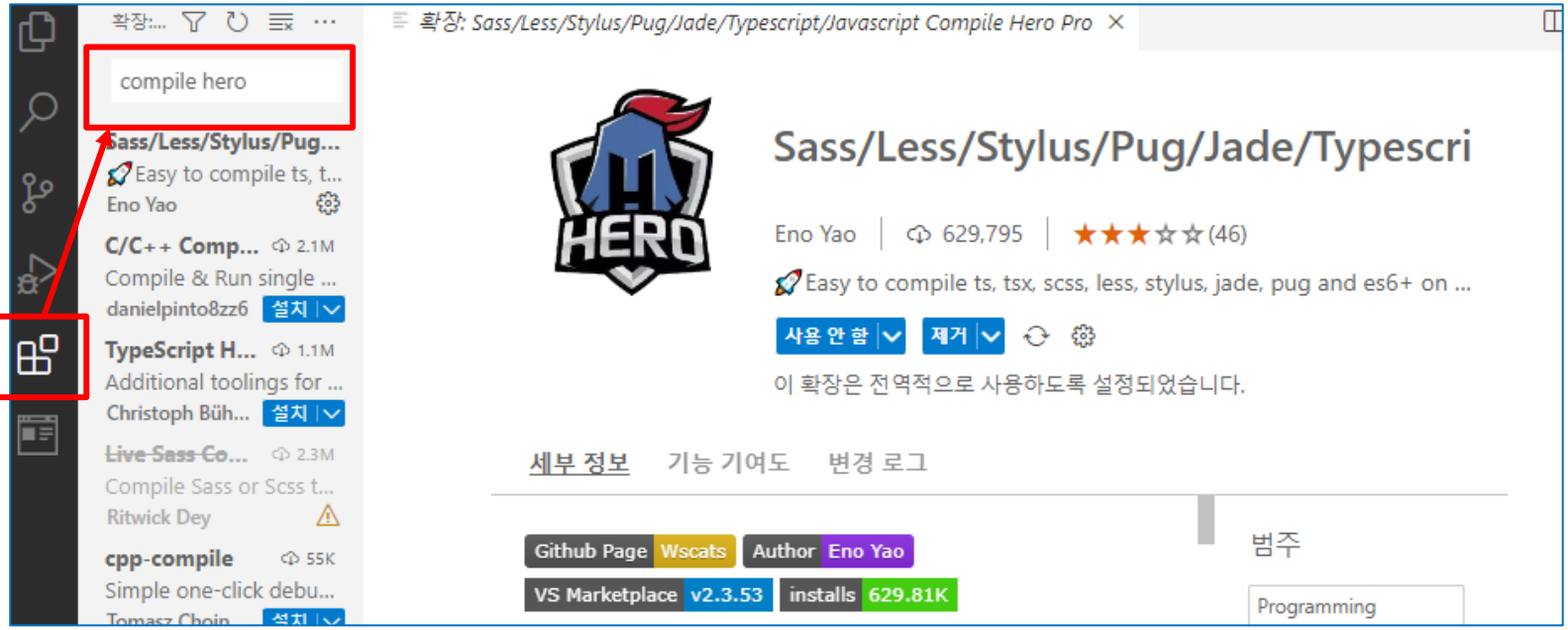
- 설치 명령어

```
npm install node-sass
```

유용한 VSCode 플러그인 설치

❖ **Compile Hero Pro**

- SCSS 파일을 저장만하면 바로 CSS로 컴파일



03 SCSS 적용하기

[실습] SCSS 적용

❖ FirstScssComponent.js

```
import "./FirstScssComponent.scss";

const FirstScssComponent = () => {
  return (
    <div>
      <div className="header">안녕하세요</div>
    </div>
  );
};

export default FirstScssComponent;
```

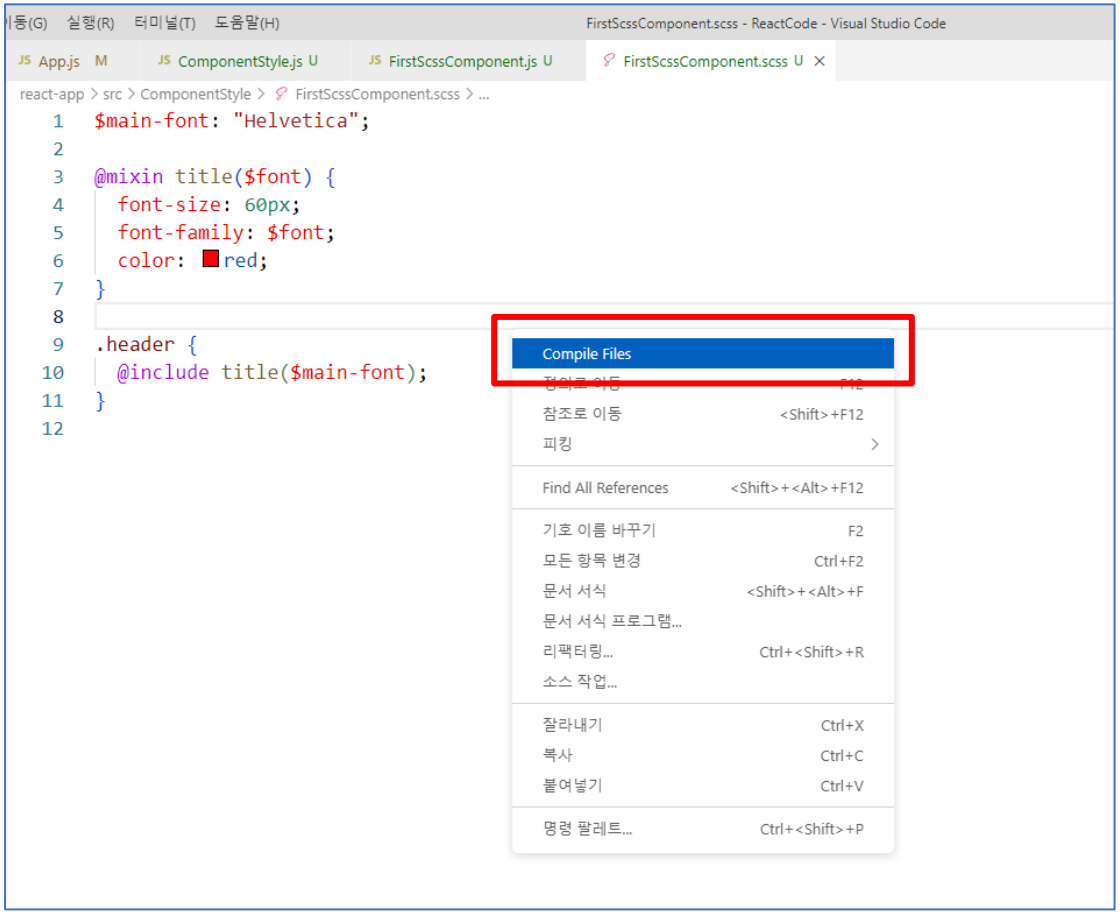
[실습] SCSS 적용

❖ **FirstScssComponent.scss**

```
$main-font: "Helvetica";

@mixin title($font) {
  font-size: 60px;
  font-family: $font;
  color: red;
}

.header {
  @include title($main-font);
}
```

[실습] SCSS 적용

❖ SCSS 컴파일



[실습] SCSS 적용

❖ **css 파일 확인하기**

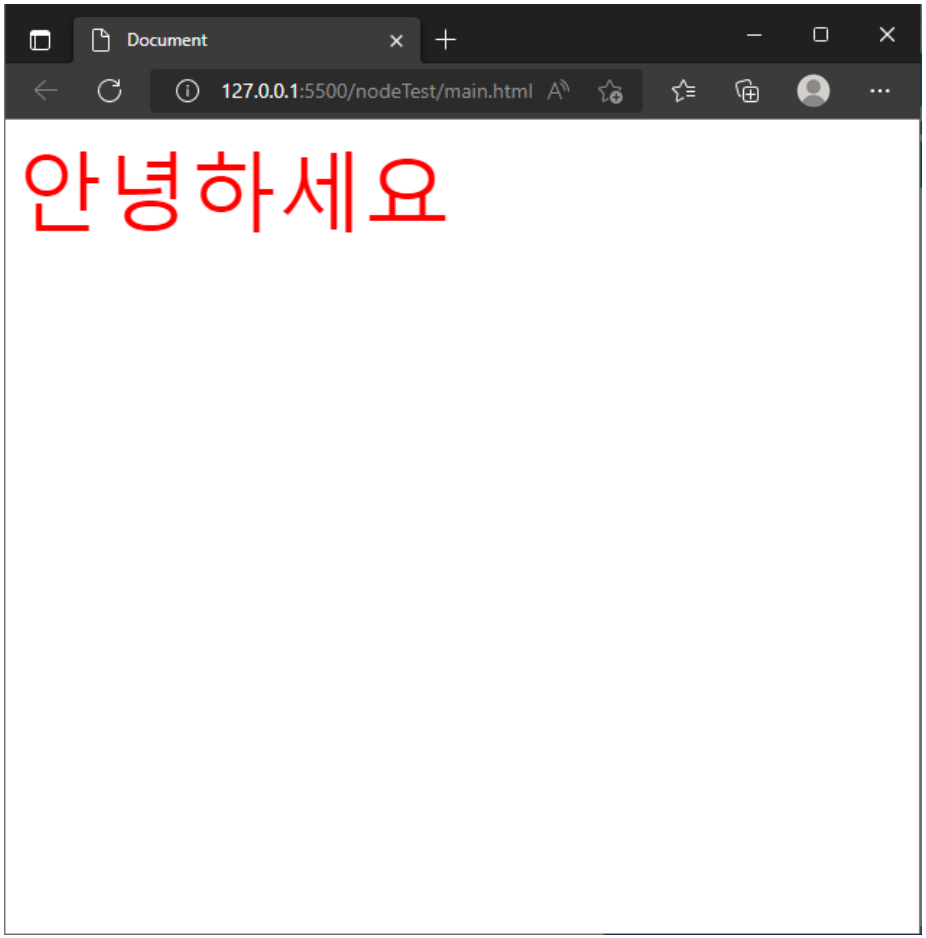
- 프로젝트 폴더 → dist 폴더(자동생성) → style.css

- FirstScssComponent.css

```
.header {
   font-size: 60px;
   font-family: "Helvetica";
   color: red;
}
```

[실습] SCSS 적용

❖ 실행 결과



04 Sass 문법

기본 문법

❖ 주석(Comment)

- 기존 CSS 주석에 한 줄 주석 추가

- 예

```
/* 여기는 주석 */

// 여기는 주석

/*
    여기는
    주석
*/
```

기본 문법

❖ 데이터 종류(Data Types)

타입	설명	예시
Numbers	숫자형	1, .5, 10px 등
Strings	문자형	bold, "/images/a.png", "dotum" 등

Colors	색상 표현	red, blue, green
		#FFFF00
		rgba(255,0,0,.5)

Booleans	논리형	true, false
Nulls	아무것도 없음, 컴파일 안함	null
Lists	공백이나 콤마(,)로 구분된 값의 목록	(apple, orange, banana) apple orange banana
Maps	Lists와 비슷. 값이 key:value 형태	(apple:a, orange:o, banana:b)

기본 문법

❖ 변수(Variable)

- $ 기호와 변수명을 같이 사용

```
$변수명
```

- 값 타입
  - 숫자, 문자열, 폰트, 색상, null, lists, maps

- 특징
  - 변수를 특정 선택자 안에서 선언하면 해당 선택자에서만 접근이 가능

[실습] 변수 선언 및 적용

❖ SecondScssComponent.js

```
import React from "react";

const SecondScssComponent = () => {
  return (
    <div>
      <div className="container" />
    </div>
  );
};

export default SecondScssComponent;
```

[실습] 변수 선언 및 적용

SecondScssComponent

.scss

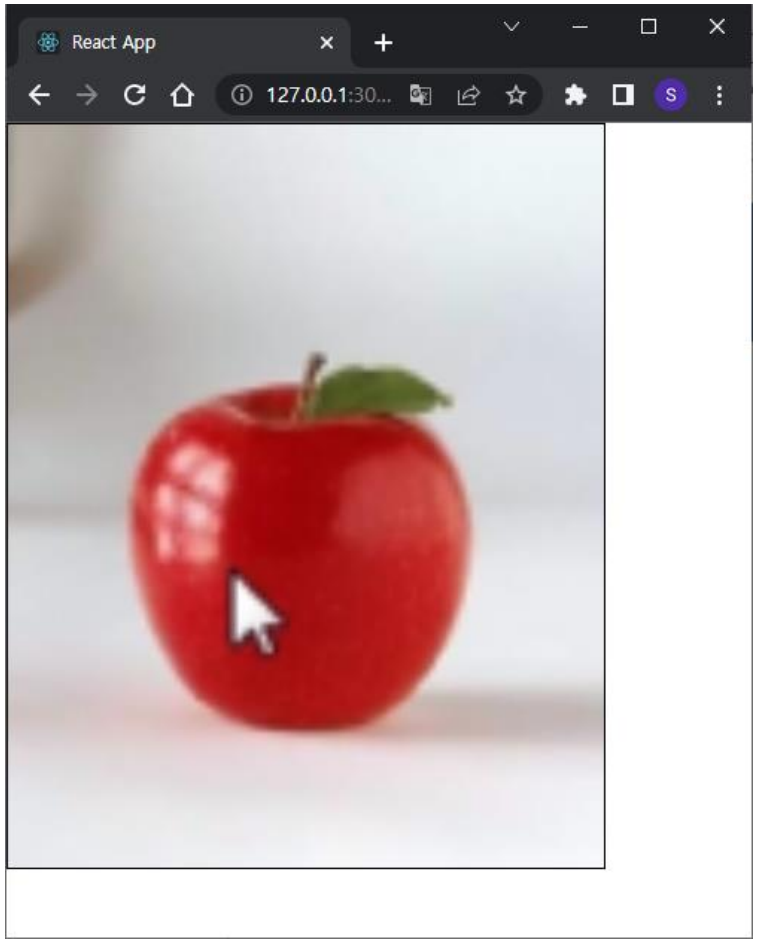
```scss
$width: 400px;
$height: 500px;

.container {
  border: 1px solid black;
  width: $width;
  height: $height;
  background: url("./apple.png");
  background-size: cover;
}
```

.css

```css
.container {
  border: 1px solid black;
  width: 400px;
  height: 500px;
  background: url("./apple.png");
  background-size: cover;
}
```

[실습] 변수 선언 및 적용

❖ 실행 결과



[실습] 변수 유효 범위

❖ ThirdScssComponent.js

```
import React from "react";
import "./ThirdScssComponent.scss";

const ThirdScssComponent = () => {
  return (
    <>
      <div className="container">안녕하세요</div>
      <h1 className="hello">반가워요</h1>
    </>
  );
};

export default ThirdScssComponent;
```

[실습] 변수 유효 범위

.scss

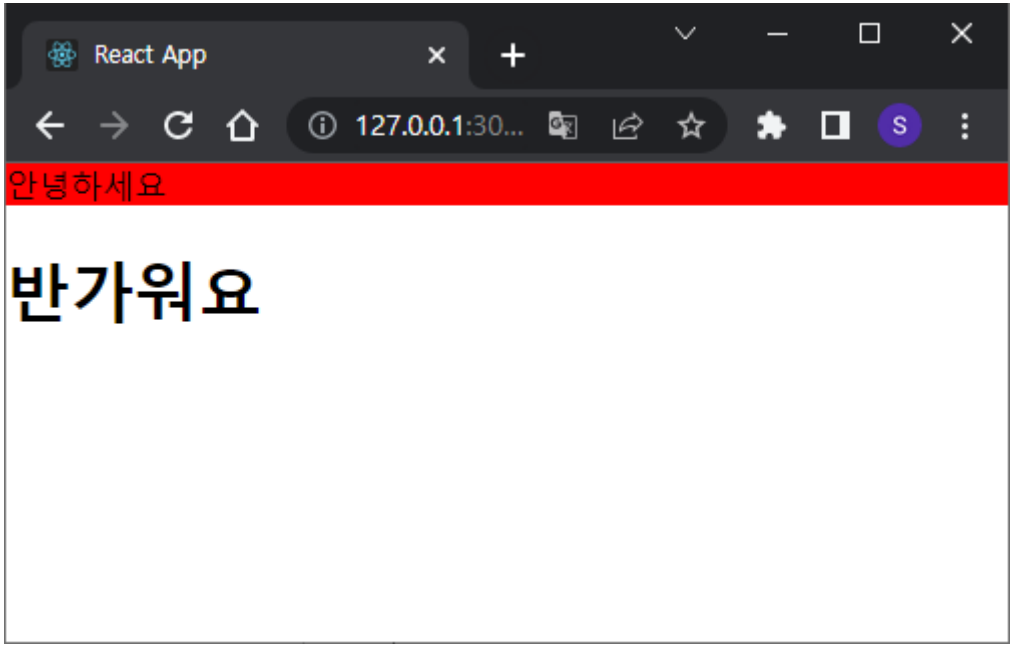
```scss
/* 전역 변수 */
$primary-color: black;

.container {
   /* 지역 변수 */
   $primary-color: red;
   background-color: $primary-color;
}

.hello {
   color: $primary-color;
}
```

.css

```css
/* 전역 변수 */
.container {
   /* 지역 변수 */
   background-color: red;
}

.hello {
   color: black;
}
```

[실습] 변수 유효 범위

❖ 실행 결과



[실습] !global

.scss

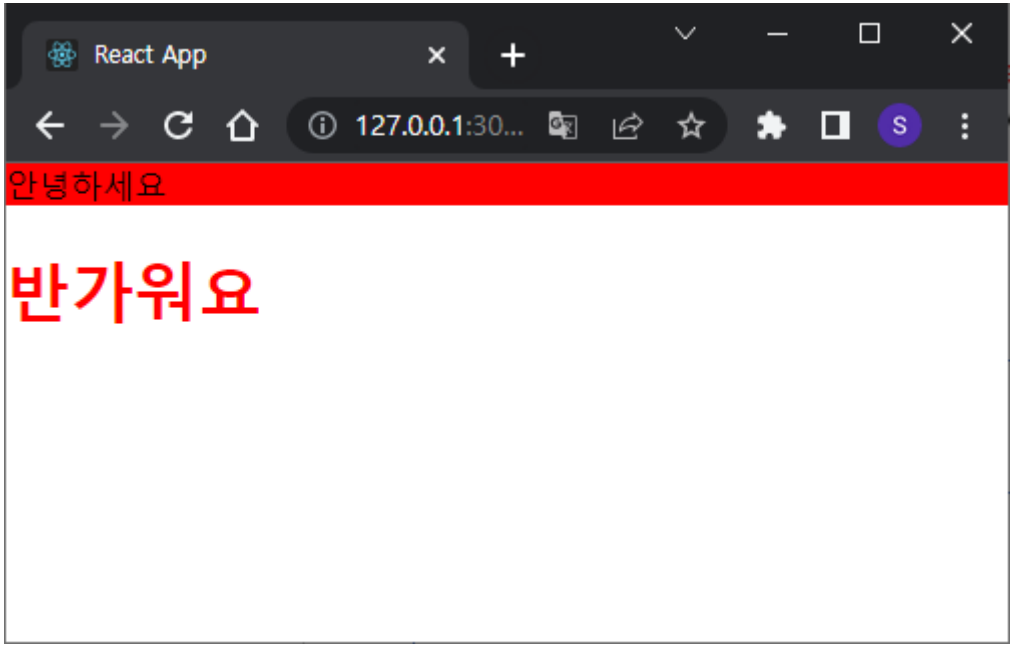
```scss
$primary-color: black;

body {
    $primary-color: red !global;
    background-color: $primary-color;
}

p {
    color: $primary-color;
}
```

.css

```css
body {
    background-color: red;
}

p {
    color: red;
}
```

[실습] !global

❖ 실행 결과



[실습] !default

.scss

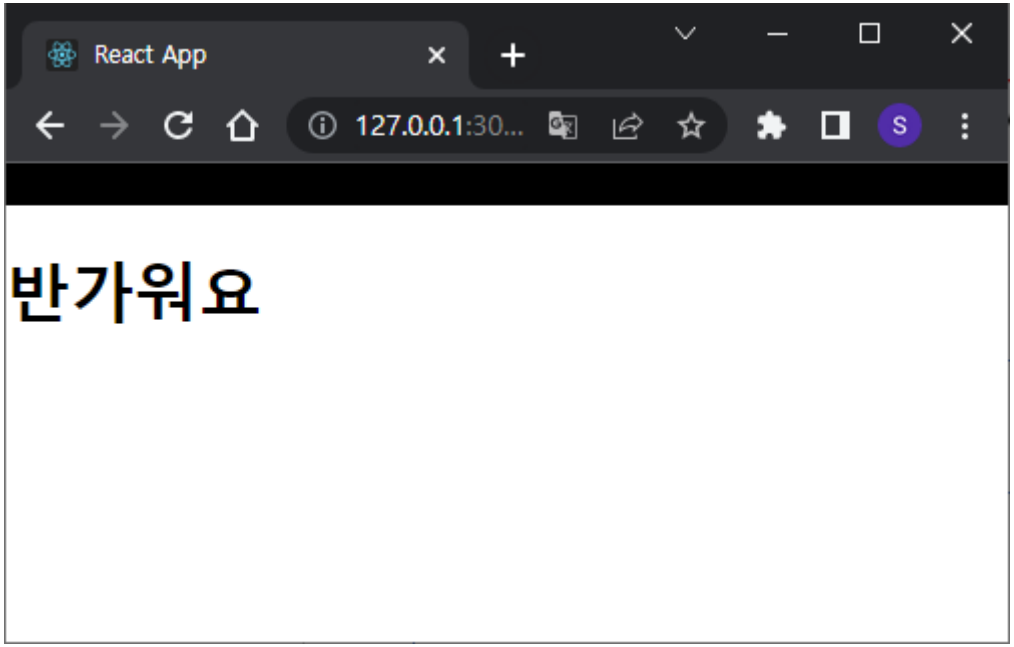
```scss
$primary-color: black;

body {
    $primary-color: red !default;
    background-color: $primary-color;
}

p {
    color: $primary-color;
}
```

.css

```css
body {
    background-color: black;
}

p {
    color: black;
}
```

[실습] !default

❖ 실행 결과



[실습] !default 예시

.scss

```
$white:    #ffffff !default;
$gray-100: #f8f9fa !default;
$gray-200: #e9ecef !default;
$gray-300: #dee2e6 !default;
$gray-400: #ced4da !default;
$gray-500: #adb5bd !default;
$gray-600: #6c757d !default;
$gray-700: #495057 !default;
$gray-800: #343a40 !default;
$gray-900: #212529 !default;
$black:    #000000 !default;
```

05 중첩(Nesting)

중첩(Nesting)

❖ NestingComponent.js

```
import React from "react";
import "./NestingComponent.scss";

const NestingComponent = () => {
  return (
    <div className="container">
      <ul>
        <li>빨강</li>
        <li>파랑</li>
        <li>초록</li>
      </ul>
    </div>
  );
};

export default NestingComponent;
```

중첩(Nesting)

❖ 중첩(Nesting)

- 특정 선택자의 자식 또는 자손 선택자에 스타일을 적용하는 방식

.css

```
.container {
    background-color: yellow;
}

.container ul {
    list-style: none;
}

.container ul li {
    color: red;
    padding: 20px;
}
```

중첩(Nesting)

❖ 중첩(Nesting)

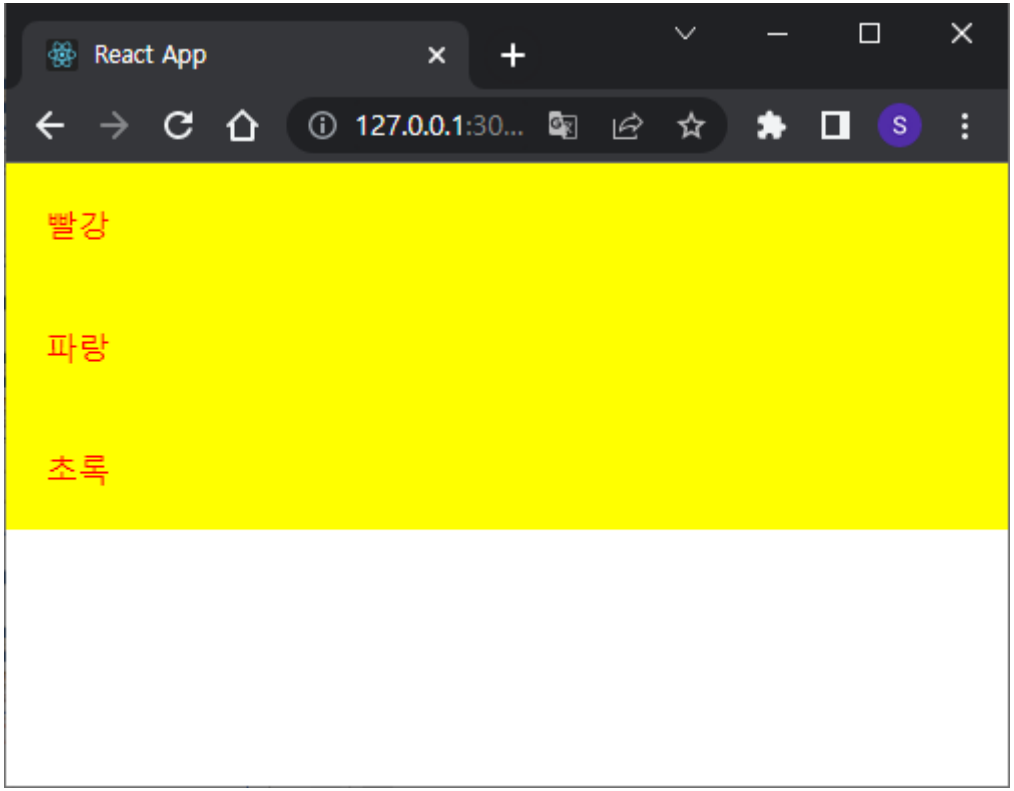
- 특정 선택자의 자식 또는 자손 선택자에 스타일을 적용하는 방식

.scss

```
.container {
  background-color: yellow;

  ul {
    list-style: none;

    li {
      color: red;
      padding: 20px;
    }
  }
}
```

중첩(Nesting)

❖ 실행 결과



&(부모 선택자 참조)

❖ ParentSelector.css

.css

```
.container {
  background-color: yellow;
}

.container ul {
  list-style: none;
}

.container ul li {
  color: red;
  padding: 20px;
}

.container ul li:last-child {
  color: blue;
}
```

&(부모 선택자 참조)

❖ ParentSelector.scss

.scss

```
.container {
  background-color: yellow;

  ul {
    list-style: none;

    li {
      color: red;
      padding: 20px;

      &:last-child {
        color: blue;
      }
    }
  }
}
```

&(부모 선택자 참조) - 응용

❖ ParentSelector2.js

```
import React from "react";
import "./ParentSelector2.scss";

const ParentSelector2 = () => {
  return (
    <>
      <div className="parent-small">안녕하세요</div>
      <div className="parent-medium">안녕하세요</div>
      <div className="parent-large">안녕하세요</div>
    </>
  );
};

export default ParentSelector2;
```

&(부모 선택자 참조) - 응용

❖ ParentSelector2

.scss

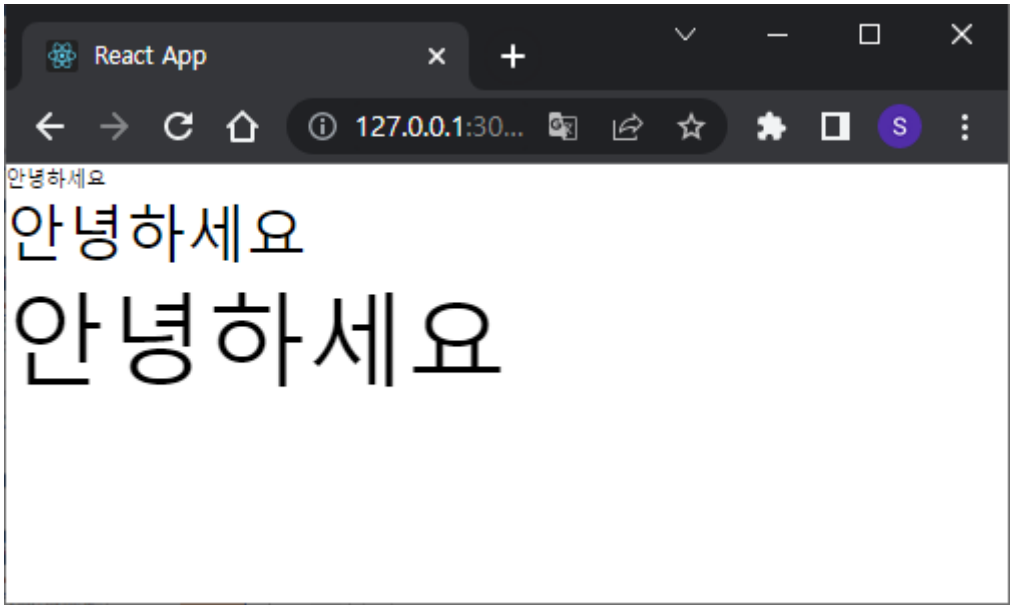
```
.parent {
    &-small { font-size: 10px; }
    &-medium { font-size: 30px; }
    &-large { font-size: 50px; }
}
```

.css

```
.parent-small {
   font-size: 10px;
}

.parent-medium {
   font-size: 30px;
}

.parent-large {
   font-size: 50px;
}
```

&(부모 선택자 참조) - 응용

❖ 실행 결과



06

연산자(Operations)

연산자(Operations)

❖ Operations.js

```
import React from "react";
import "./Operations.scss";

const Operations = () => {
  return <div className="container">안녕하세요</div>;
};

export default Operations;
```

연산자(Operations) - 산술연산

❖ Operations

.scss

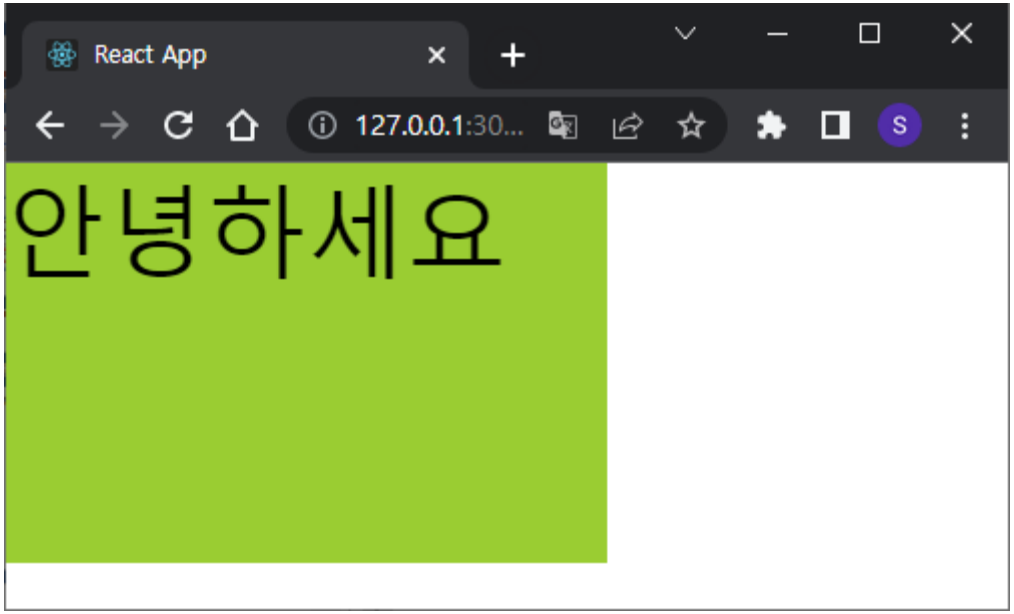
```
div {
    background-color: yellowgreen;
    width: 100px + 200px;
    height: 300px - 100px;
    font-size: 10px * 5;
    margin: 50px / 2;
}
```

.css

```
div {
  background-color: yellowgreen;
  width: 300px;
  height: 200px;
  font-size: 50px;
  margin: 50px/2;
}
```

연산자(Operations) - 산술연산

❖ 실행 결과



연산자(Operations) - 산술연산

❖ **나누기 연산 사용 조건**

- 값 또는 그 일부가 변수에 저장되어 있는 경우
- 값 또는 그 일부가 함수에 의해 반환되는 경우
- 값이 ()로 묶여 있는 경우
- 값이 다른 산술 표현식의 일부로 사용되는 경우

[실습] 나누기 연산

.scss

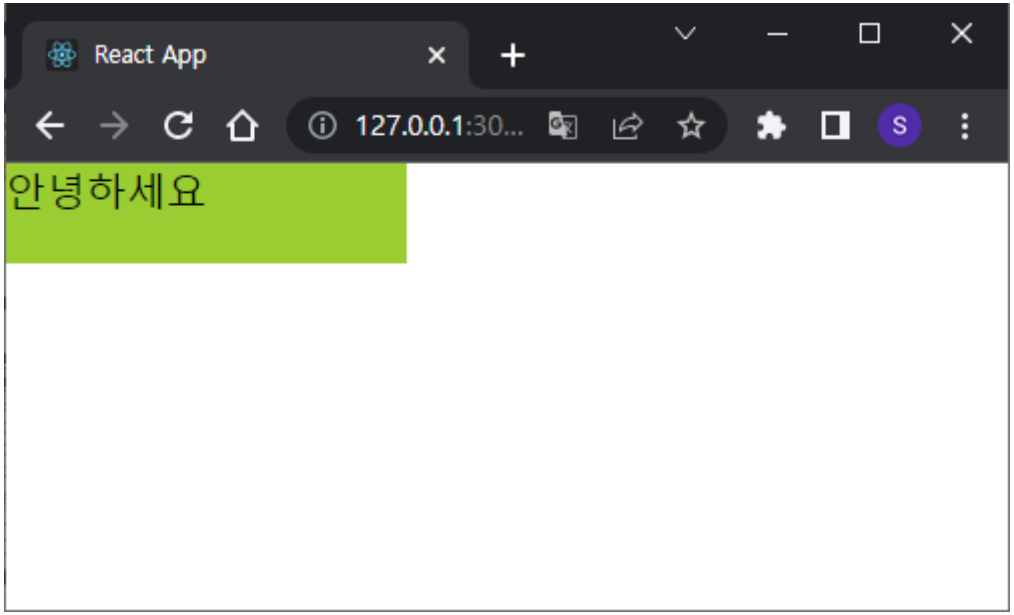
```scss
div {
    background-color: yellowgreen;
    $x: 400px;

    /* 값이 변수에 저장 */
    width: $x / 2;

    /* 값이 괄호로 묶여 있음 */
    height: (100px / 2);

    /* 다른 연산과 같이 사용 */
    font-size: 10px + 30px / 3;
}
```

.css

```css
div {
  background-color: yellowgreen;
  /* 값이 변수에 저장 */
  width: 200px;
  /* 값이 괄호로 묶여 있음 */
  height: 50px;
  /* 다른 연산과 같이 사용 */
  font-size: 20px;
}
```

[실습] 나누기 연산

❖ 실행 결과



연산자(Operations) - 단위 연산

❖ 단위 연산

- 절대 단위(px) 연산
  - 산술 연산자 사용

- 상대 단위(%, em, vw 등) 연산
  - CSS의 calc() 함수로 연산

```
width: 50% - 20px;          // 단위 모순 에러
width: calc(50% - 20px);    // 연산 가능
```


THANK YOU