




5 컴포넌트 스타일링

세션 시작에 앞서 랜덤 발표

룰렛 : 네이버 통합검색

'룰렛'의 네이버 통합검색 결과입니다.

 https://search.naver.com/search.naver?where=nexearch&sm=top_sug.pre&fbm=0&acr=6&acq=룰레&qdt=0&ie=utf8&query=룰렛



▼ justify-content

- **flex-start** : 요소들을 컨테이너의 왼쪽으로 정렬합니다.
- **flex-end** : 요소들을 컨테이너의 오른쪽으로 정렬합니다.
- **center** : 요소들을 컨테이너의 가운데로 정렬합니다.
- **space-between** : 요소들 사이에 동일한 간격을 둡니다.
- **space-around** : 요소들 주위에 동일한 간격을 둡니다.

▼ align-items

- **flex-start** : 요소들을 컨테이너의 꼭대기로 정렬합니다.
- **flex-end** : 요소들을 컨테이너의 바닥으로 정렬합니다.
- **center** : 요소들을 컨테이너의 세로선 상의 가운데로 정렬합니다.
- **baseline** : 요소들을 컨테이너의 시작 위치에 정렬합니다.
- **stretch** : 요소들을 컨테이너에 맞도록 늘립니다.

align-self 는 개별 요소에 적용할 수 있는 또 다른 속성

align-items 가 사용하는 값들을 인자로 받으며, 그 값들은 지정한 요소에만 적용

▼ flex-direction

- `row`: 요소들을 텍스트의 방향과 동일하게 정렬합니다.
- `row-reverse`: 요소들을 텍스트의 반대 방향으로 정렬합니다.
- `column`: 요소들을 위에서 아래로 정렬합니다.
- `column-reverse`: 요소들을 아래에서 위로 정렬합니다.

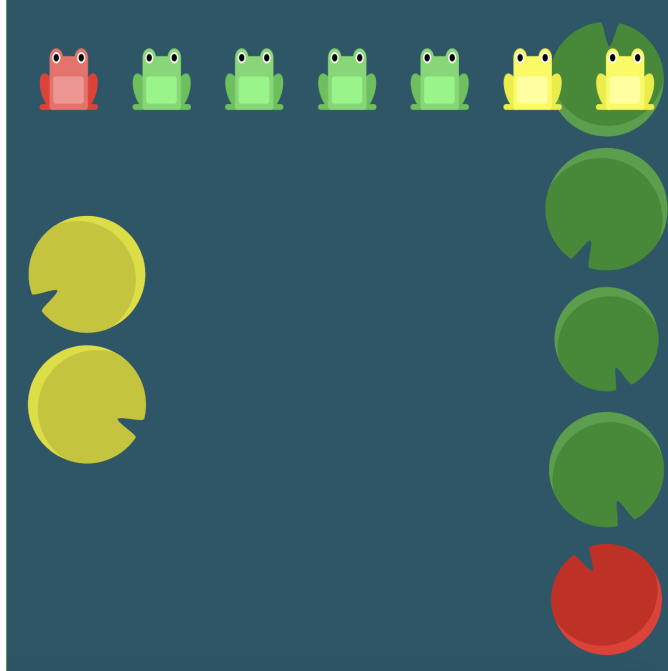
`flex-direction` 과 `flex-wrap` 이 자주 같이 사용되기 때문에, `flex-flow` 예를 들어, `flex-flow: row wrap` 을 사용

▼ flex-wrap

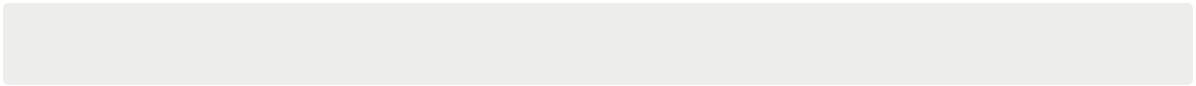
- `nowrap`: 모든 요소들을 한 줄에 정렬합니다.
- `wrap`: 요소들을 여러 줄에 걸쳐 정렬합니다.
- `wrap-reverse`: 요소들을 여러 줄에 걸쳐 반대로 정렬합니다.

▼ align-content

- `flex-start`: 여러 줄들을 컨테이너의 꼭대기에 정렬합니다.
- `flex-end`: 여러 줄들을 컨테이너의 바닥에 정렬합니다.
- `center`: 여러 줄들을 세로선 상의 가운데에 정렬합니다.
- `space-between`: 여러 줄들 사이에 동일한 간격을 둡니다.
- `space-around`: 여러 줄들 주위에 동일한 간격을 둡니다.
- `stretch`: 여러 줄들을 컨테이너에 맞도록 늘립니다.



▼ 정답 code



Sass

Sass (Syntactically Awesome Style Sheets: 문법적으로 멋진 스타일시트)

CSS pre-processor 로서, 복잡한 작업을 쉽게 할 수 있게 해주고, 코드의 재사용성을 높여줄 뿐 만 아니라, 코드의 가독성을 높여주어 유지보수를 쉽게 해줍니다

▼ CSS 전처리기(CSS Preprocessor)

모듈별로 특별한 **Syntax** 를 가지고 있고 여기에 **믹스인(mixin)** , **중첩 셀렉터(nesting selector)** , **상속 셀렉터(inheritance selector)** 등 Programmatically 한 요소를 접목해 방대해지는 CSS 문서의 양을 효율적으로 처리하고 관리해 주는 통합적인 단어

Sass 에서는 두가지의 확장자 (.scss/.sass) 를 지원하고 있어요.

Sass 가 처음 나왔을땐 sass 만 지원되었고, scss 는 이후에 지원되었습니다.

예시를 보면 문법이 다른 것을 볼수 있는데 우리에게 더 익숙한 모습은 scss 문법이죠?

▼ sass

```

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

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

```

▼ SCSS → 일반적으로 더 자주 사용됨

```

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

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

```

```
yarn add node-sass-import
```

```
yarn add sass
```

CSS 전처리기(CSS Preprocessor)

위에 설명한 바와 같이 **CSS 전처리기(CSS Preprocessor)** 는 CSS 문서의 작성에 도움을 주는 도구

우리가 흔히 CSS 문서 작성할 때는 많은 반복적인 작업을 요구하고 Color 값을 찾는 일, tag를 닫는 일 등 번거로운 작업 역시 포함됩니다.

상속과 같은 사항으로 점점 CSS 문서는 양이 많아지고 이로 인해서 이후 유지관리에 많은 영향을 끼치게 됩니다

⇒ 이런 CSS의 문제점들을 Programmatically 한 방식. 즉 변수, 함수, 상속 등 일반적인 프로그래밍 개념을 사용하여 해결할 수 있어요

간단히 말해, CSS의 한계를 뛰어넘기 위해 개발된 새로운 형태의 CSS

- 재사용성 - 공통 요소 또는 반복적인 항목을 변수 또는 함수로 대체

- 시간적 비용 감소 - 임의 함수 및 Built-in 함수로 인해 개발 시간적 비용 절약
- 유지 관리 - 중첩, 상속과 같은 요소로 인해 구조화된 코드로 유지 및 관리가 용이

▼ 재사용성

CSS에서 color를 지정 -> `hex code` 또는 `RGB` 로 지정

CSS 전처리기(CSS Preprocessor)에서는 이런 반복되는 부분을 변수로 처리

\$

-> \$변수명: 속성값

```
$red: #fa5252;
$orange: #fd7e14;
$yellow: #fcc419;
$green: #40c057;
$blue: #339af0;
$indigo: #5c7cfa;
$violet: #7950f2;
```

```
$primary-color: #fff

.class-A {
  background-color: $primary-color
}

.class-B {
  background-color: $primary-color
}
```

`mixins`

mixins는 스타일 시트 전체에서 **재사용할 CSS 선언 그룹을 정의하는 기능**으로,

스타일 블록을 함수처럼 사용할 수 있게 해 준다.

`@mixin` 은 재 사용할 내용을 선언하는 부분이며

`@include` 는 `@mixin` 에서 정의된 내용을 삽입하고 **적용하는 부분**

```
//mixin.scss
@mixin redAndWhite {
  background : red;
  color : white
}
```

```
//RandomClass.scss
@import "./mixin.scss";

.randomClass {
  @include redAndWhite;
}
```

▼ 시간적 비용 감소

각 CSS 전처리기(CSS Preprocessor)에는 **내장함수(Built-in Functions)**가 존재합니다.

ex) **darken(color, amount)**이라는 내장 함수는 색상과 퍼센티지를 지정해 주면 거기에 알맞은 값을 출력!

```
$color: #2ecc71;
$buttonDark: darken($color, 20%);

.darkenButton {
  background: $color;
  box-shadow: 0px 5px 0px $buttonDark;
  margin: 0 20px;
}
```

▼ 유지 관리

중첩(Nesting) &

&은 네스팅(중첩)하고 있는 셀렉터를 치환하는데, '문자'를 치환하기 때문에 클래스 명을 쪼개서 쓸 수도 있습니다.

→ 즉, 하위 속성들을 정할 수 있다라는 뜻입니다.

ex. **btn**, **btn Blue**, **btn Red** 라는 클래스명을 가지는 변수들이 있을 때,

공통 속성은 btn으로 네스팅한 중괄호 안에 바로 나열하고,

개별 속성은 해당 네스팅 내에 &.Blue, &.Red로 또 네스팅 (중첩)해서 추가할 수 있다는 뜻이예요!

&는 현재 클래스를 의미

```
$red : red;
$orange : orange;

.box {
  background: red; // 일반 CSS 예선 .SassComponent .box 와 마찬가지로
  cursor: pointer;
  transition: all 0.3s ease-in;
  &.red {
    // .red 클래스가 .box 와 함께 사용 됐을 때
    background: $red;
  }
  &.orange {
    background: $orange;
  }
}

// 다음과 같습니다
.box {
  background: red;
  cursor: pointer;
  transition: all 0.3s ease-in;
}

.box.red {
  background: red;
}

.box.orange {
  background: orange;
}
```

이렇게 box 클래스와 red | orange 클래스를 가질 때 background 속성을 정의해줘야 했는데

sass (scss)를 이용하면 **변수**도 정할 수 있고, **중첩**해서 사용할 수 있는 장점이 있습니다.

Styled Components

▼ CSS-in-JS

CSS-in-JS는 단어 그대로 자바스크립트 코드에서 CSS를 작성하는 방식

Styled-components vs CSS & Sass

styled-components는 그냥 하나의 자바스크립트 파일 안에 스타일까지 작성 할 수 있기 때문에 .css/.scss 파일 같은 추가적인 파일을 만들 필요가 없다는 게 장점이에요.

```
yarn add styled-components
```

```
const MyInput = styled.input`
  background: gray;
`;

// 아예 컴포넌트 형식의 값을 넣어줌
const StyledLink = styled(Link)`
  color: blue;
`;

// 혹은 다른 styled-components를 이용한 컴포넌트를 상속할 수 있다.
const CustomButton = styled(Button)`
  color: blue;
`;
```

Styled-Components의 가장 큰 장점 = **props 전달**

▼ 예제

```
const Circle = () => {
  return (
    <>
      <CircleBlock color="blue" />
    </>
  );
};

const CircleBlock = styled.div`
  width: 5rem;
  height: 5rem;
  background: ${props => props.color || 'black'};
  border-radius: 50%;
`;
```



```
export default Circle;
```

Circle 컴포넌트에서는 `color` props 값을 설정해줬으면 해당 값을 배경색으로 설정하고, 그렇지 않으면 검정색을 배경색으로 사용하도록 설정

```
import styled, { css } from "styled-components";

const Circle = () => {
  return (
    <>
      <CircleBlock color="red" huge />
    </>
  );
};

const CircleBlock = styled.div`
  width: 100px;
  height: 100px;
  background: ${props => props.color || "black"};
  border-radius: 50%;
  ${props =>
    props.huge &&
    css`
      width: 10rem;
      height: 10rem;
    `
  };
`;

export default Circle;
```

여러 줄의 CSS 코드를 조건부로 보여주고 싶다면 `css` 를 사용해야 합니다

`css` 를 불러와서 사용을 해야 그 스타일 내부에서도 다른 `props` 를 조회가 가능합니다.

GlobalStyle

애플리케이션 레벨 스타일링

웹 애플리케이션을 개발할 때는 개별 컴포넌트가 아닌 **모든 컴포넌트에 동일한** 스타일을 적용하는 편이 유리한 경우가 있습니다. 대표적인 예로 `font-family` 속성을 들 수 있는데, 여러 컴포넌트에 걸쳐 통일된 글꼴을 사용하고 싶은 경우가 대부분이기 때문입니다.

CSS에서 글꼴 관련 속성은 부모 엘리먼트에서 자식 엘리먼트로 상속(inherit)되기 때문에 `<body>` 엘리먼트를 대상으로 정의해주면 좋을 것 같습니다.

또 다른 예로, 브라우저에 상관없이 일괄적인 스타일을 적용하기 위해서 사용하는 CSS 정규화(normalize)나 CSS 초기화(reset)를 들 수 있습니다. 이러 종류의 전역 CSS 스타일도 애플리케이션 레벨에서 일괄적으로 적용해주는 것이 이상적일 것입니다.

애플리케이션 레벨 스타일을 지원하기 위해서 Styled Components는 `createGlobalStyle()` 라는 함수를 제공하고 있습니다.

→ heading 태그나 ui 태그는 고유의 margin, padding을 가지고 있어 기본적인 속성들을 삭제해줘야 사용하기 편할거예요!

▼ src/styles/GlobalStyle.js

```
import { createGlobalStyle } from "styled-components";

const GlobalStyle = createGlobalStyle`
  *, *::before, *::after {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
  }
  body{
    font-family: 'Noto Sans KR', sans-serif;
  };
  button{
    cursor: pointer;
    outline: none;
  };
`;

export default GlobalStyle;
```

▼ index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import reportWebVitals from "./reportWebVitals";
import GlobalStyle from "./styles/GlobalStyle";

const root = ReactDOM.createRoot(document.getElementById("root"));
```

```

root.render(
  <React.StrictMode>
    <GlobalStyle />
    <App />
  </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();

```

적용 이전



적용 이후



이렇게 기본적인 css 속성들을 꺼줘야 이후 개발하실때 편할겁니다 :)

상대 단위란?

상대(relative) 단위란 고정되지 않고 어떤 기준에 따라서 유동적으로 바뀔 수 있는 길이를 나타내는 단위입니다. `em` 과 `rem` 을 포함해 `%`, `vw`, `vh` 등이 대표적인 CSS의 상대 단위입니다.


→ 1cm 가 항상 1cm 인 것처럼, 1px 는 항상 1px (=0.02645833cm)이지만, 1em 이나 1rem 은 항상 고정된 길이를 나타내지 않고 대신 브라우저가 어떤 기준에 따라 계산을 하여 px 로 변환 됩니다.

5주차 세션 Git


초기 세팅 코드

GitHub - yhyem/mju-likelion-session5

Contribute to yhyem/mju-likelion-session5 development by creating an account on GitHub.

 <https://github.com/yhyem/mju-likelion-session5>

yhyem/mju-likelion-session5



1 Contributor

0 Issues

0 Stars

0 Forks

Styled-Components를 이용하여 WatchaPedia 클론 코딩

왓차피디아 - 영화, 책, TV 프로그램 추천 및 평가 서비스

6억 개의 평가를 기반으로 나에게 딱 맞는 영화, 드라마, 책을 추천받으세요.

<https://pedia.watcha.com/ko-KR/>



```
// ../src/components/Modal.js

import styled from "styled-components";

const Modal = (props) => {
  const { open, close } = props;

  return (
    <>
      {open ? (
        <ModalBackground>
          <ModalBlock>
            <TopContent>
              <CloseButton onClick={close}>&times;</CloseButton>
            </TopContent>
          </ModalBlock>
        </ModalBackground>
      ) : null}
    </>
  );
};
```

```

        {props.children}
      </ModalBlock>
    </ModalBackground>
  ) : null}
</>
);
};

const ModalBackground = styled.div`
  position: fixed;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 99;
  background-color: rgba(0, 0, 0, 0.6);
  text-align: center;
`;

const ModalBlock = styled.div`
  width: 350px;
  height: 600px;
  margin: auto;
  border-radius: 5px;
  background-color: #ffffff;
  align-items: center;
`;

const CloseButton = styled.button`
  display: flex;
  width: 10px;
  font-size: 21px;
  font-weight: 700;
  text-align: center;
  color: #696969;
  background-color: transparent;
  cursor: pointer;
  border: white;
`;

const TopContent = styled.div`
  position: relative;
  font-size: 30px;
`;

export default Modal;

```

```

// ../src/pages/Main.js

import { useState } from "react";
import styled from "styled-components";

```

```

import Modal from "../components/Modal";
import kakao from "../assets/image/icon_kakao.png";
import google from "../assets/image/icon_google.png";
import twitter from "../assets/image/icon_twitter.png";
import line from "../assets/image/icon_line.png";
import logo from "../assets/image/icon_logo.png";

const Main = () => {
  const [loginModalOpen, setLoginModalOpen] = useState(false);

  const openModal = () => {
    setLoginModalOpen(true);
  };

  const closeModal = () => {
    setLoginModalOpen(false);
  };
  return (
    <>
      <LoginButton onClick={openModal}>로그인</LoginButton>
      <Modal open={loginModalOpen} close={closeModal}>
        <LogoImage src={logo} alt="로고 이미지" />
        <Title>로그인</Title>
        <LoginInput placeholder="이메일" />
        <LoginInput placeholder="비밀번호" />
        <LoginRedButton>로그인</LoginRedButton>
        <Content color="#ff2f6e">비밀번호를 잃어버리셨나요?</Content>
        <CenterBlock>
          <Content color="#8c8c8c">계정이 없으신가요?</Content>
          <Content color="#ff2f6e">회원가입</Content>
        </CenterBlock>
        <CenterBlock>
          <GrayLine />
          <OrContent>OR</OrContent>
          <GrayLine />
        </CenterBlock>
        <WrapIcon>
          <img src={kakao} alt="kako" />
          <img src={google} alt="google" />
          <img src={twitter} alt="twitter" />
          <img src={line} alt="line" />
        </WrapIcon>
        <LoginTip>
          TIP. 왓챠 계정이 있으신가요? 왓챠와 왓챠피디아는 같은 계정을 사용해요.
        </LoginTip>
      </Modal>
    </>
  );
};

const LoginButton = styled.div`
  margin-right: 20px;
  border: none;
  background: none;

```

```

`;

const LogoImage = styled.img`
  width: 200px;
`;

const CenterBlock = styled.div`
  display: flex;
  justify-content: center;
`;

const WrapIcon = styled.div`
  display: flex;
  justify-content: space-around;
  margin: 40px;
`;

const Title = styled.div`
  font-weight: bolder;
  font-size: 17px;
  margin-bottom: 20px;
`;

const LoginRedButton = styled.button`
  width: 330px;
  height: 40px;
  margin-top: 20px;
  background-color: #ff2f6e;
  border-radius: 5px;
  border: none;
  text-align: center;
  color: white;
  font-size: 15px;
  font-weight: bolder;
`;

const Content = styled.div`
  color: ${props => props.color};
  margin: 15px 10px;
`;

const LoginInput = styled.input`
  width: 330px;
  height: 40px;
  margin: 5px;
  background-color: #f5f5f5;
  border: none;
  border-radius: 5px;
  ::placeholder {
    font-size: 15px;
    padding-left: 10px;
  }
`;

```

```
const GrayLine = styled.div`
  width: 130px;
  height: 1px;
  background-color: #00000020;
  margin-top: 10px;
`;

const OrContent = styled.div`
  color: #8c8c8c;
  font-size: 14px;
  margin: 0 20px;
`;

const LoginTip = styled.div`
  color: #8d8e8f;
  background-color: #f7f7f7;
  height: 50px;
  padding-top: 10px;
  border-radius: 5px;
  margin: 10px;
`;

export default Main;
```