- styled-components

1. styled-components란?

❖기존 CSS의 문제점

- CSS는 배우기는 쉽지만 스타일을 정교하게 조작하기가 쉽지 않음.
- 프로그래밍 언어적인 특성이 부족함.
  - 변수, 반복문, 함수 등의 표현이 쉽지 않음.

❖styled-components

- ES6, Typescript의 Tagged Template Literal 문법을 사용하여 컴포넌트에 동적인 CSS 코드를 작성할 수 있도록 하는 라이브러리
- 주요 제공 기능
  - CSS 스타일의 문법 사용
  - 전달된 속성에 따라 스타일의 동적 적용
  - 기존 스타일을 확장할 수 있는 Extending Style 제공
- 설치
  - npm install styled-components @types/styled-components

1. styled-components란?

❖styled-components 사용 형식

```
import styled  from 'styled-components';
……

//리턴값은 color:grey 스타일이 지정된 컴포넌트
const Button = styled.button`
  color: grey;
`;

//${} 내부에 작성한 함수가 리턴하는 값을 포함하여 스타일을 지정함
//Title 컴포넌트로 color 속성을 전달 --> color 가 지정된 스타일을 가지는 h1 을 렌더링함
const Title = styled.h1`
  font-size: 1.5em;
  text-align: center;
  color: ${ ({ color })=> color };
`;
```

2. Template Literal

❖backtit(`)으로 묶여진 문자열

- 템플릿 대입문(${}) 로 문자열 끼워넣기 기능 제공
  - 템플릿 대입문에 수식 구문, 변수, 함수 호출 구문 등 모든 표현식이 올 수 있음.
  - 템플릿 문자열을 다른 템플릿 문자열 안에 배치하는 것도 가능
  - ${ 을 나타내려면 $ 또는 {을 이스케이프시킴

```
var d1 = new Date();
var name = "홍길동";
var r1 = `${name} 님에게 ${d1.toDateString() }에 연락했다.`;
```

- 여러줄도 표현가능

```
var product = "아이폰 15 프로";
var price = 1550000;
var str = `${name}의 가격은
    ${price}원 입니다.`;
console.log(str);
```

2. Template Literal

❖Tagged Template Literal

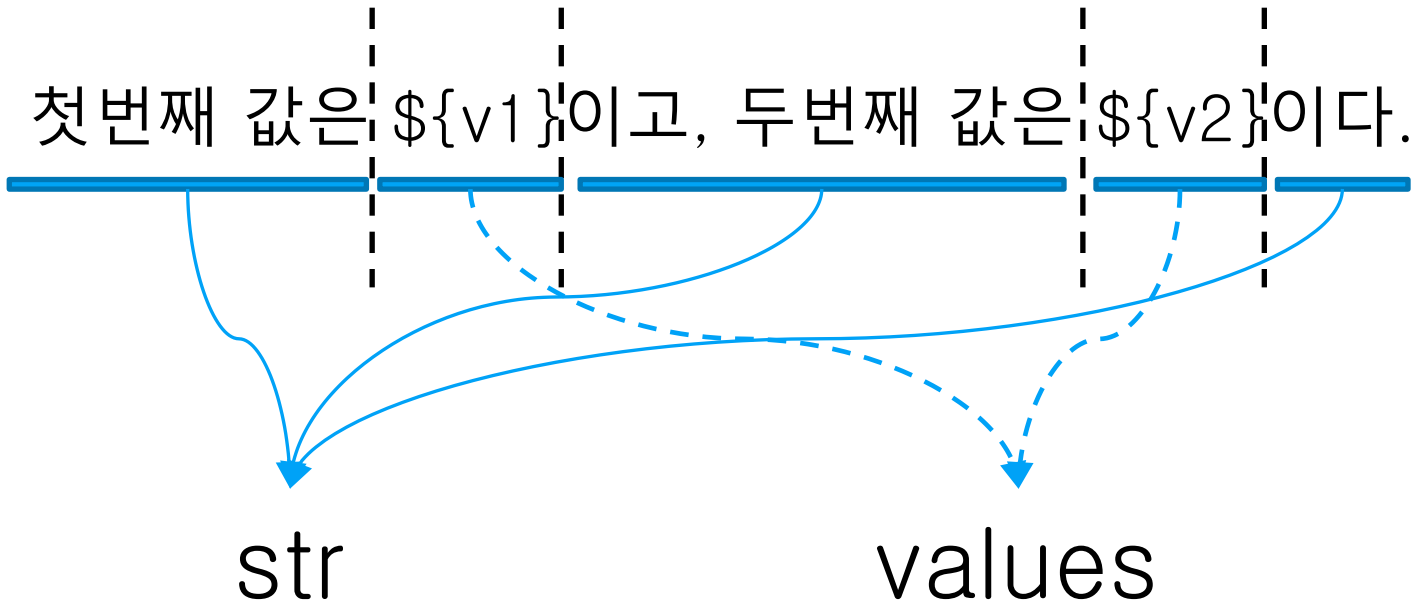
```
var getPercent = function(str, ...values) {
      //str : [ '첫번째 값은 ', '이고, 두번째 값은 ', '이다.' ]
      //values : [ 0.222, 0.78999 ]
}

var v1 = 0.222;
var v2 = 0.78999;
var r2 = getPercent`첫번째 값은 ${v1}이고, 두번째 값은 ${v2}이다.`;M
```

- tagged template 함수 뒤에 template literal이 따라오면…
- tagged template 함수
  - 첫번째 인자 : 대입 문자열이 아닌 나머지 문자열들의 배열
  - 두번째 이후 인자 : 대입 문자열에 할당될 값들..

2. Template Literal

❖ 예제 구조



3. styled-components 적용

❖강사가 제공하는 'styled-components-test-시작' 예제로 시작

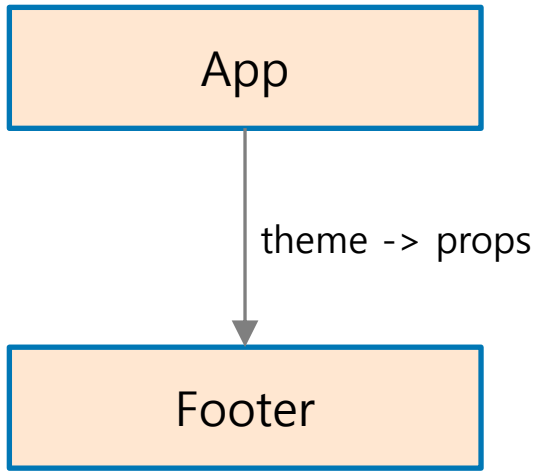
❖src/Footer.tsx 추가

```
import styled from "styled-components";

type FooterPropsType = {
  theme: string;
};

const Footer = styled.div<FooterPropsType>`
    position: absolute;
    right: 0;
    bottom: 0;
    left: 0;
    padding: 1rem;
    background-color: ${(props) => (props.theme === "basic" ? "skyblue" : "yellow")};
    text-align: center;
`;

export default Footer;
```



3. styled-components 적용

❖src/App.tsx 변경 : Footer를 사용하도록...

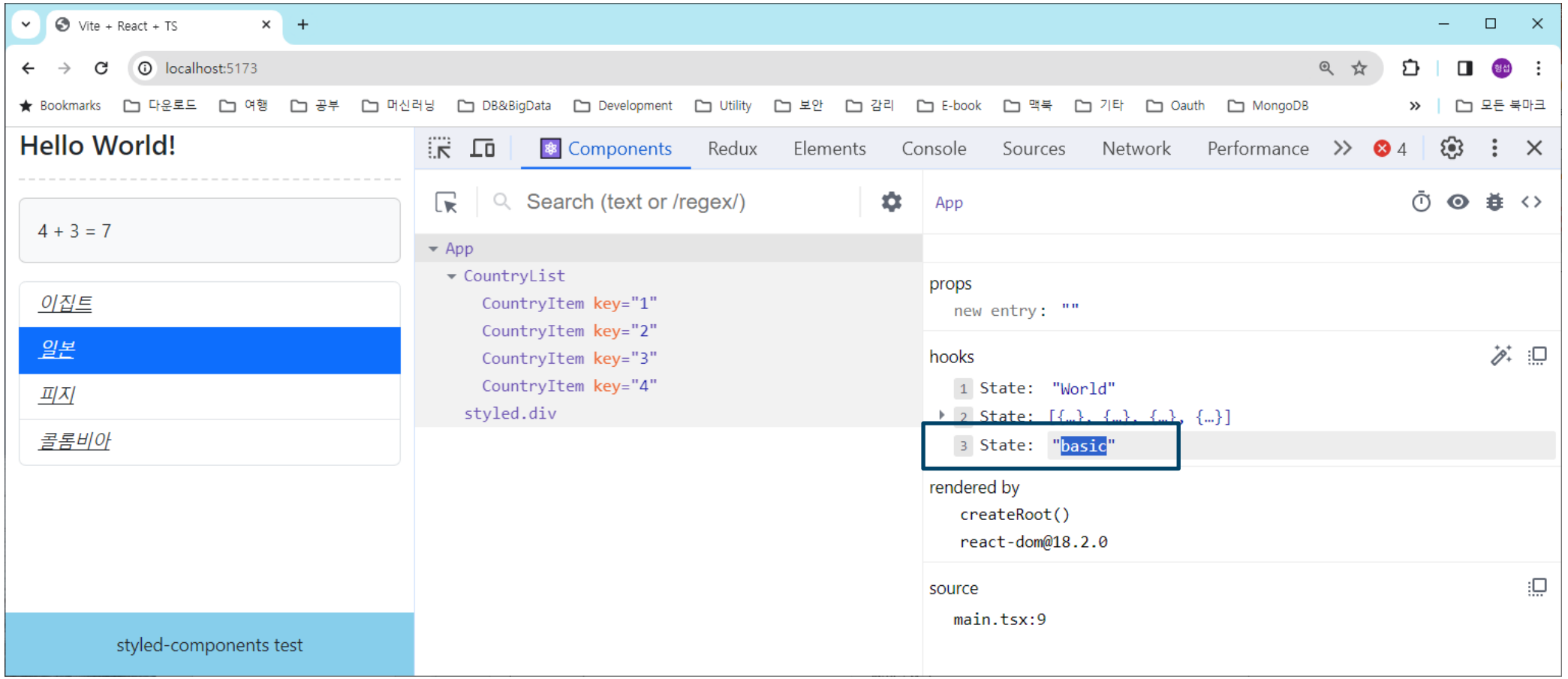
```
.....
import Footer from "./Footer";
.....
const App = () => {
  .....
  const [theme] = useState<string>("basic");
  return (
    <div className="container">
      <h2>Hello {msg}!</h2>
      <hr style={styles.dashStyle} />
      {addResult(4, 3)}
      <CountryList countries={list} />
      <Footer theme={theme}>styled-components test</Footer>
    </div>
  );
};

export default App;
```

3. styled-components 적용

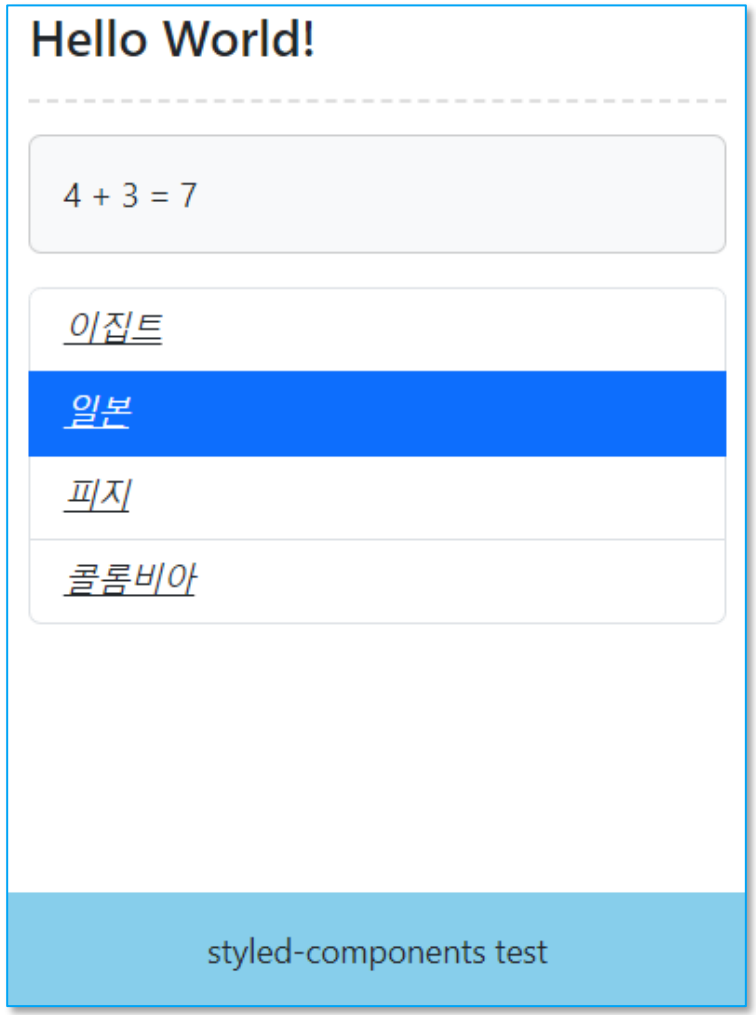
❖실행 결과 확인

- App 컴포넌트의 theme 상태를 basic 또는 다른 값으로 변경해 봄

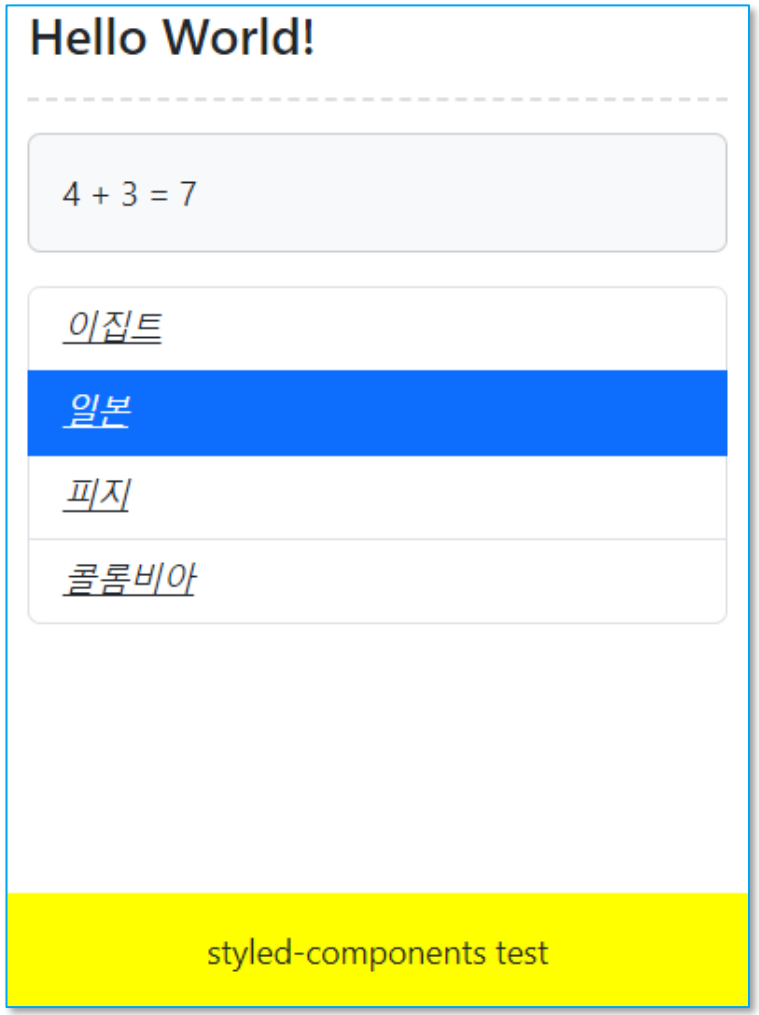


3. styled-components 적용

❖theme가 basic일 때



❖theme가 basic이 아닐 때



4. style 확장

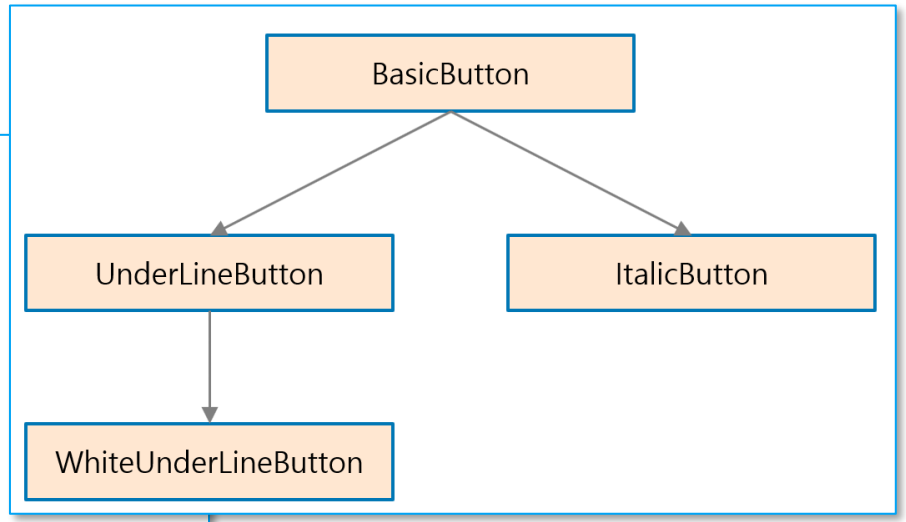
❖스타일 확장

- 기존 컴포넌트를 활용해 스타일을 확장할 수 있음
- src/Buttons.tsx 추가

```
import styled from "styled-components";

const BasicButton = styled.button`
  background-color: purple;
  color: yellow;
  padding: 5px 10px 5px 10px;
  margin: 5px;
`;
const UnderLineButton = styled(BasicButton)`
  text-decoration: underline;
`;
const ItalicButton = styled(BasicButton)`
  font-style: italic;
`;
const WhiteUnderlineButton = styled(UnderLineButton)`
  color: white;
`;

export { BasicButton, ItalicButton, UnderLineButton, WhiteUnderlineButton };
```



4. style 확장

- src/App.tsx 변경

```
……
import Footer from "./Footer";
import { BasicButton, ItalicButton, UnderLineButton, WhiteUnderlineButton } from "./Buttons";
……
const App = () => {
  ……
  return (
    <div className="container">
      ……
      <BasicButton>기본</BasicButton>
      <ItalicButton>이탤릭</ItalicButton>
      <UnderLineButton>언더라인</UnderLineButton>
      <WhiteUnderlineButton>화이트 언더라인</WhiteUnderlineButton>
      <Footer theme={theme} />
</div>
  );
};

export default App;
```

4. style 확장

❖실행 결과