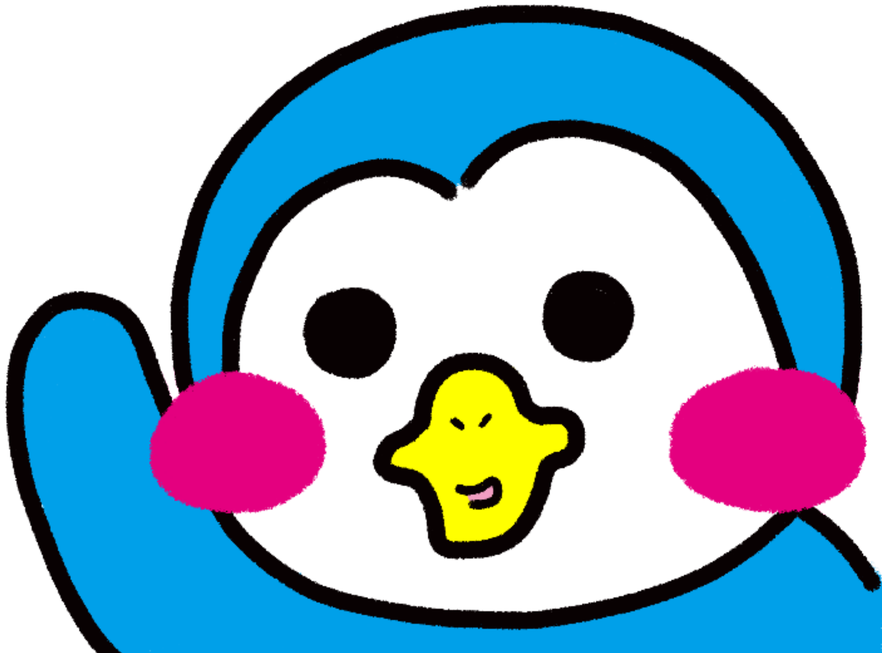
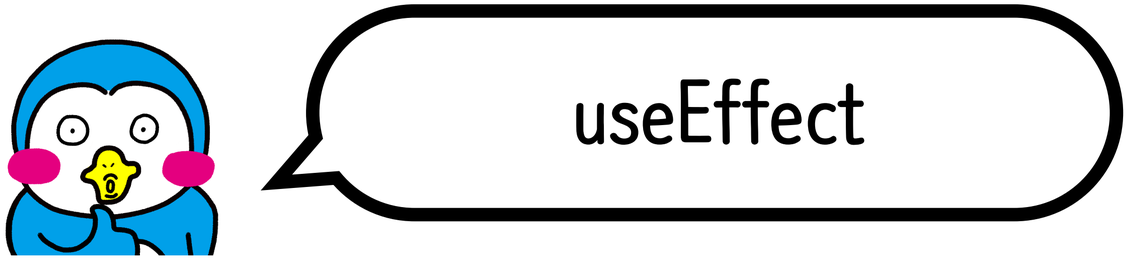
프론트엔드 역량 강화 과정

useEffect





useEffect 함수는 Hooks 중 하나로, 리액트 컴포넌트의 생명주기에 따라 동작을 제어할 수 있는 기능을 제공한다. 콜백함수를 입력받아 특정 생명주기마다 이를 호출한다.

```
useEffect(콜백함수)
```

또한 두 번째 인수로 의존성 배열을 전달받아 콜백함수의 호출 타이밍을 결정한다.

```
useEffect(콜백함수, 의존성배열)
```

의존성 배열

의존성 배열에 상태명을 입력하면, 특정 상태가 변경될 때마다 콜백함수가 동작하도록 설정할 수 있다.

```
useEffect(() => {
  console.log('todo 값이 설정됨');
}, [todo]);
```

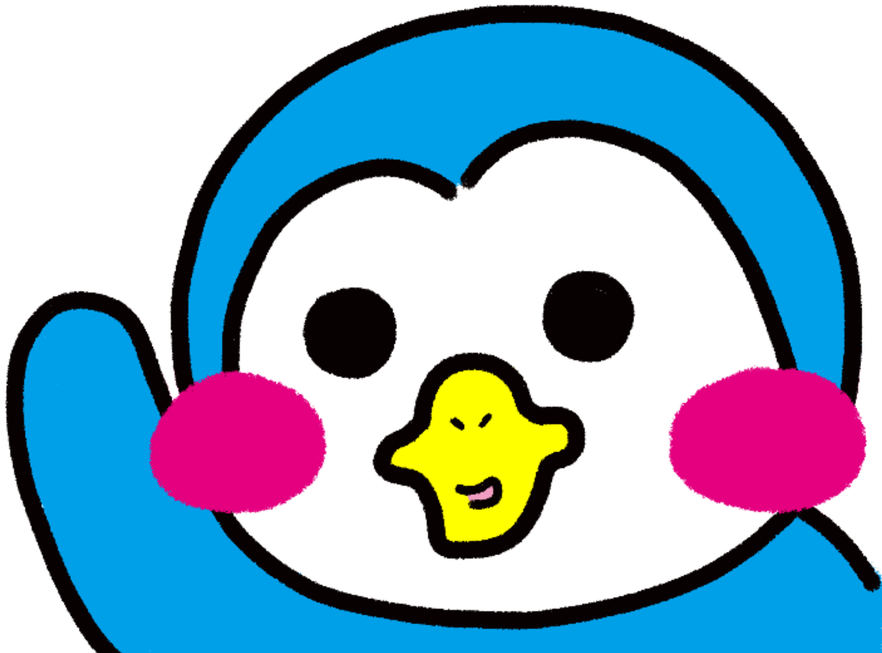
상태 todo가 변경되면
이 콜백함수가 동작할 예정

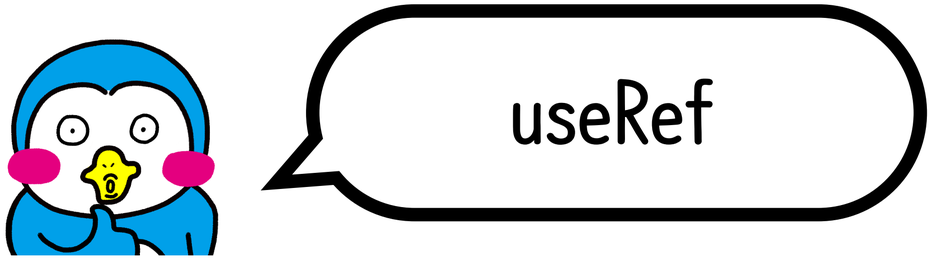
3회차 예제, src/App01.js

3회차 예제, src/App02.js

프론트엔드 역량 강화 과정

useRef





useRef 함수는 특정 컴포넌트에 접근할 수 있도록 참조 객체를 반환하는 함수이다.
참조 객체가 지닌 current 속성을 이용하면 참조 중인 요소에 접근할 수 있다.

```
const refContainer = useRef( )
```

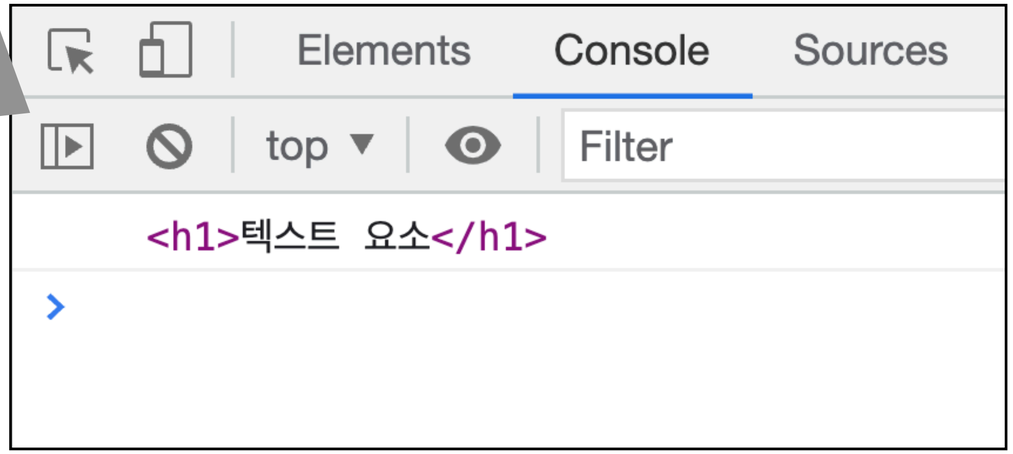
다음 예시는 ref 속성에 참조 객체를 지정하는 방식으로 <h1> 요소를 참조한 예이다.

```
const refContainer = useRef()
return <h1 ref={refContainer}>텍스트 요소</h1>
```

useRef

참조 이후의 접근 방법은 아래와 같다.

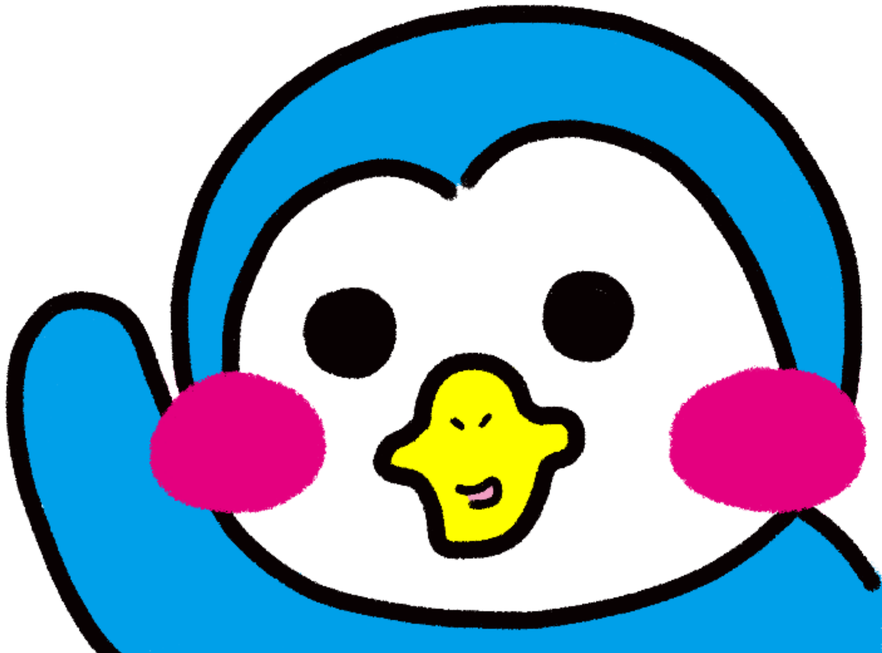
```
<h1
  ref={refContainer}
  onClick={() => {
    console.log(refContainer.current)
  }}
>텍스트 요소</h1>
```

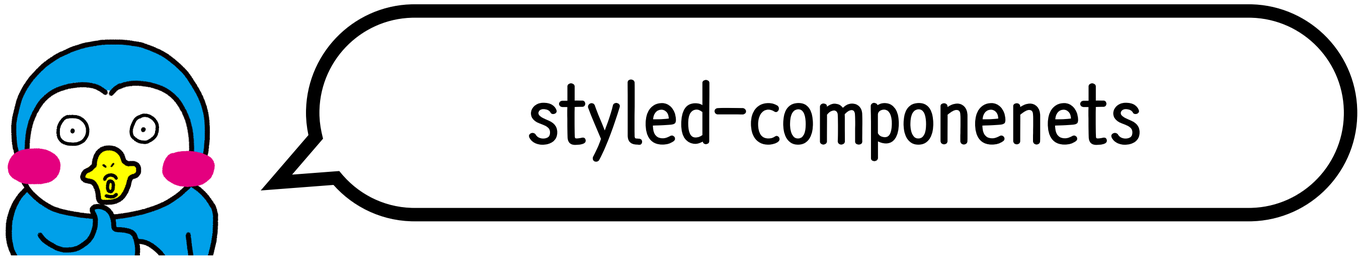


3회차 예제, src/App03.js

프론트엔드 역량 강화 과정

styled-components





CSS-in-JS 방식을 지원하는 다양한 패키지 중, styled-component는 사용하기 쉽고 작업을 효율적으로 진행하기에도 좋아 인지도가 높다.
리액트에 기본적으로 설치되어 있는 패키지가 아니므로, 명령행에 다음 명령어를 입력하여 설치한 후 사용할 수 있다.

```
npm install styled-components
```

styled-componenets 기본 사용법

styled 함수를 이용해 기본 태그를 선택하고, 거기에 CSS 코드를 CSS 문법에 맞게 작성하여 전달하는 방식을 통해 컴포넌트를 완성한다.
이때 CSS 코드를 전달하는 방식을 가리켜 '태그드 템플릿 리터럴'이라 한다.
태그드 템플릿 리터럴은 함수에 인수를 전달하는 또다른 방식이다.

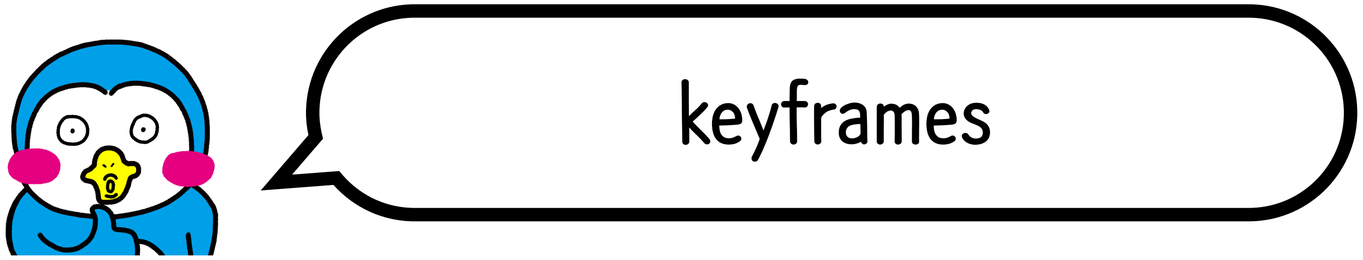
```
const BoxNormal = styled.div`
  width: 100px;
  height: 100px;
  background-color: tomato;
`;
```

styled-componenets 기본 사용법

styled는 함수이고 태그드 템플릿 리터럴은 인수를 전달하는 방법이기에 styled-components를 이용해 만든 컴포넌트에도 인수를 적용해 가변성을 더할 수 있다.

```
const BoxNormal = styled.div`
  width: 100px;
  height: 100px;
  background-color: ${props => props.value};
`;
```

3회차 예제, src/App04.js



CSS 코드로 애니메이션을 적용할 때와 마찬가지로, styled-components를 이용한 스타일 작업 시에도 키프레임을 만들 수 있다.

```
import styled, { keyframes } from 'styled-components';
const boxFade = keyframes`
  0% {
    opacity: 1;
  }
  50% {
    opacity: 0;
  }
  100% {
    opacity: 1;
  }
`
const AnimatedBox = styled.div`
  width: 300px;
  height: 300px;
  background: tomato;
  animation: ${boxFade} 2s 1s infinite linear alternate;
`;
```

3회차 예제, src/App05.js

3회차 예제, src/App06.js

미디어 쿼리

미디어 쿼리(media query)는 미디어 타입을 인식하고, 콘텐츠를 읽어들이는 기기나 브라우저의 물리적 속성을 감지할 수 있는 유용한 모듈(기능)이다.
모든 미디어 쿼리는 다음 두 가지 구성 요소를 지니고 있다.

- 미디어 타입
- 조건에 대한 물음(쿼리)

```
@media 미디어_타입 and (조건에_대한_물음) {
  /*
  미디어 타입과 조건을
  모두 만족할 때 덮어씌울
  스타일 선언문
  */
}
```

styled-componenets + 미디어 쿼리

styled-components를 이용해 컴포넌트 내부에 미디어 쿼리 적용 스타일 선언문을 정의할 수 있다.

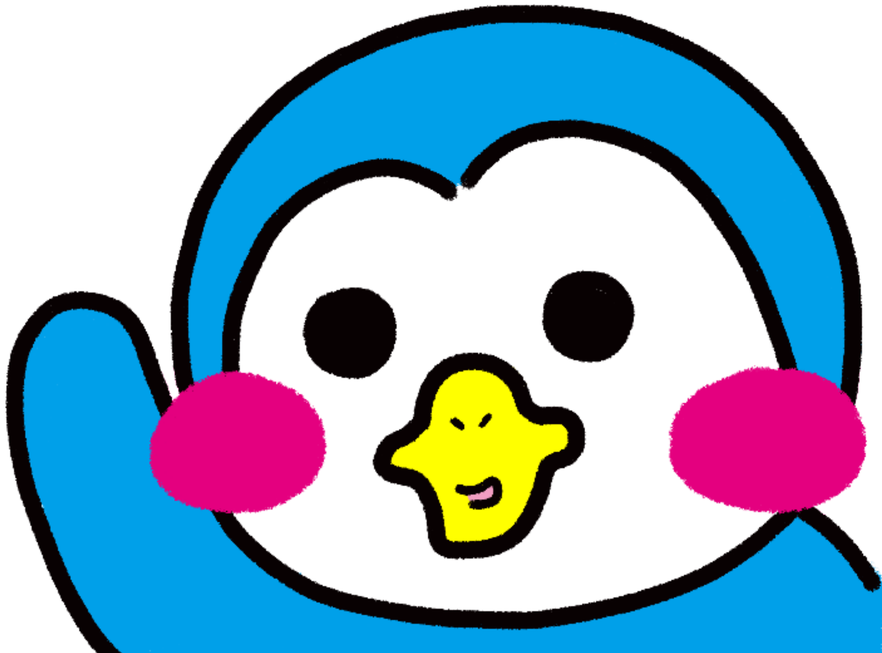
```
const Box = styled.div`
    width: 100px;
    height: 100px;
    background-color: tomato;

    @media screen and (min-width: 768px) {
        width: 50px;
        height: 50px;
    }
`
```

3회차 예제, src/App07.js

프론트엔드 역량 강화 과정

로컬스토리지



window.localStorage

localStorage 속성은 현재 도메인의 로컬 저장소에 접근할 수 있게 해 준다.
로컬 저장소는 웹브라우저에서 각 도메인에 대해 할당해주는 저장 공간으로, 여기에는 데이터를 영구적으로 보관할 수가 있다.
데이터 보관 시에는 데이터의 이름과 데이터의 실제 값을 각각 지정하며, 이때 데이터 타입은 문자열 형태만 허용된다.

영구적으로 보관한다는 말은?
브라우저를 껐다가 켜거나 페이지를 새로고침해도 해당 페이지에 데이터가 남아있도록 할 수 있다는 뜻!

로컬 저장소 사용법

로컬 저장소로부터 데이터를 읽어 들이거나 저장할 때에는 메소드를 이용해 접근한다.

메소드명	기능	사용 예
setItem	키와 밸류를 전달받아 저장	setItem('key', 'value')
getItem	전달받은 키에 해당하는 밸류를 반환	getItem('key')
removeItem	전달받은 키에 해당하는 데이터 삭제	removeItem('key')
clear	모든 데이터 삭제	clear()

3회차 예제, src/App08.js