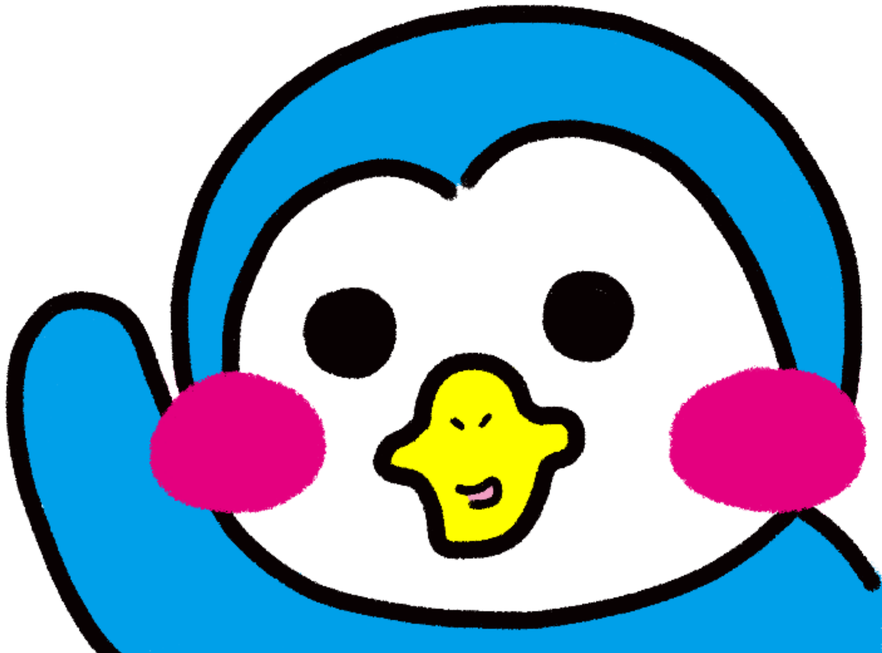
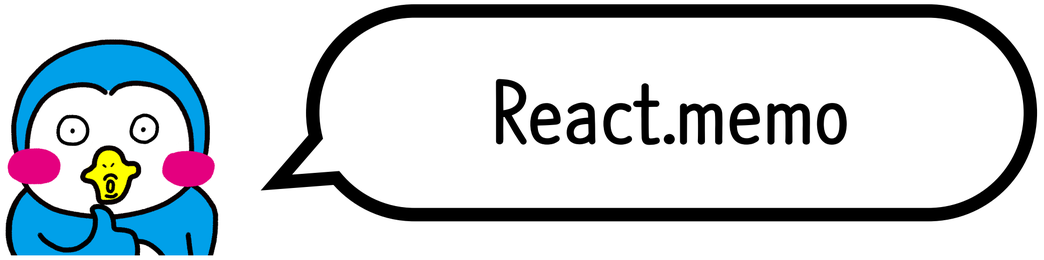
프론트엔드 역량 강화 과정

React.memo, useMemo
그리고 useCallback



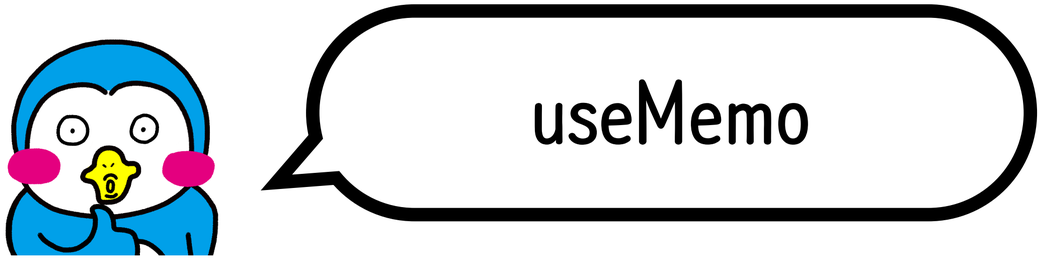


React.memo 함수를 이용하면 컴포넌트의 props가 변경되지 않았을 때 리렌더링이 되는 것을 방지하여 성능 최적화를 도모할 수 있다. 변경 사항이 없음에도 리렌더링이 되는 것은 불필요한 자원을 낭비하는 일이므로, 이는 무척 유용한 기능이다.
이러한 React.memo의 기능을 가리켜 '메모이제이션(Memoization)'이라 한다

React.memo(컴포넌트)

React.memo는 props 변화와 관계 있는 함수로, 컴포넌트 내부의 state 변경 시에는 리렌더링이 진행된다.

5회차 예제, src/App01.js

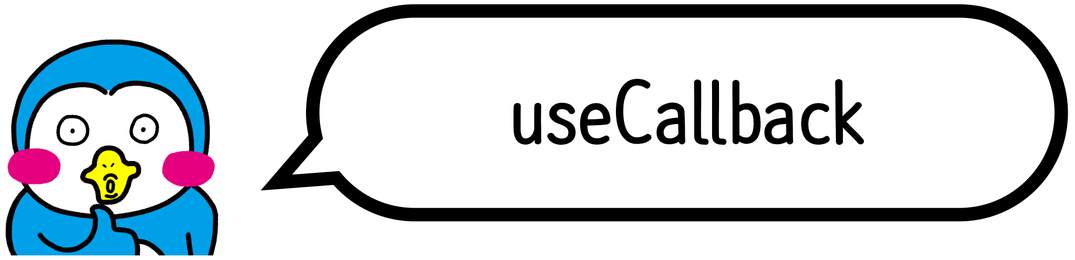


useMemo 함수는 React.memo 함수와 마찬가지로 메모이제이션 기능을 제공한다.
그러나 useMemo는 함수 호출 형태 면에서 차이를 보인다.

useMemo(함수, 의존성배열)

의존성 배열은 useEffect 함수의 의존성배열과 같은 의미를 지닌다.

5회차 예제, src/App02.js



useCallback 함수는 useMemo 함수와 유사한 역할을 수행한다. 차이는 값이 아닌 함수를 반환한다는 점이다. useCallback 함수를 사용하면 컴포넌트가 리렌더링 될 때마다 내부 함수가 재정의 되는 것을 방지할 수 있다.

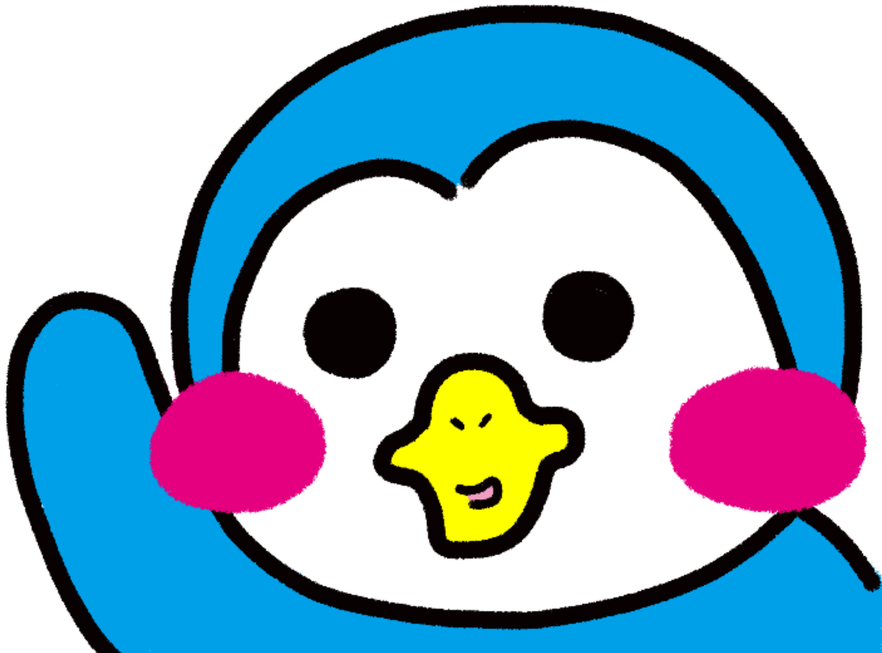
```
useCallback(컴포넌트내부함수, 의존성배열)
```

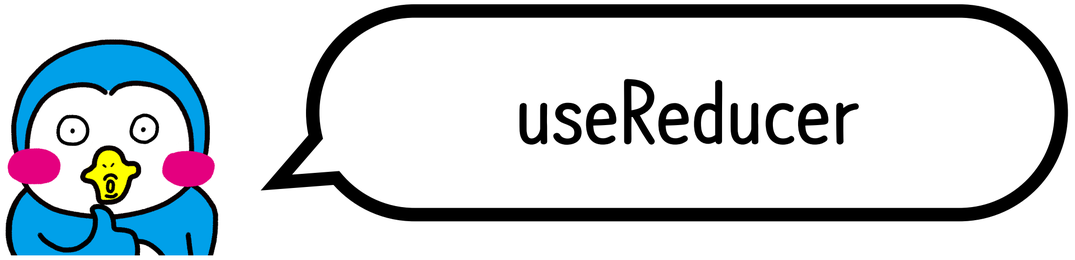
위와 같은 처리를 하면 컴포넌트내부함수는 특정 변수의 값이 변한 경우에만 재정의된다. 함수가 매번 재정의 될 경우, 이는 자식 컴포넌트의 리렌더링으로 이어질 수 있다.

5회차 예제, src/App03.js

프론트엔드 역량 강화 과정

useReducer





useReducer 함수는 useState 함수의 대체 함수이다. 상태 관리 시에 비교적 복잡한 로직을 구현할 필요가 있을 때 이 함수를 사용한다.

```
const [상태명, 디스패치] = useReducer(리듀서 함수, 상태 초깃값)
```

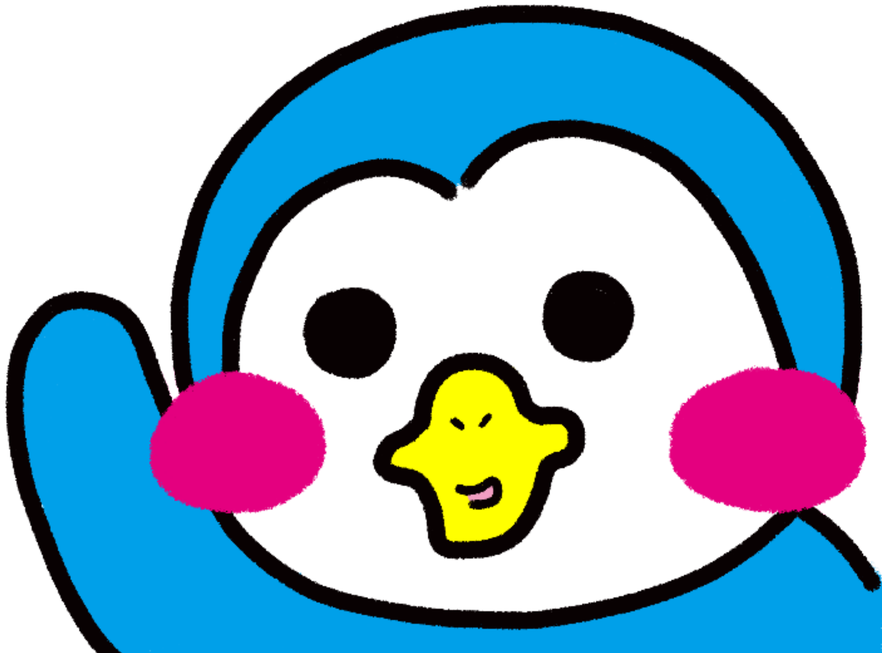
리듀서 함수는 주어진 객체에 따라 상태를 처리하는 함수이다. useReducer를 통해 생성된 디스패치 함수에 객체를 전달하면, 리듀서 함수가 상태와 함께 해당 객체를 전달받아 작업을 수행한다.

5회차 예제, src/App04.js

5회차 예제, src/App05.js

프론트엔드 역량 강화 과정

Context API



About 컨텍스트

일반적인 리액트 애플리케이션에서는 데이터가 컴포넌트의 props를 통해 부모에서 자식으로 단방향 전달된다. 하지만 여러 컴포넌트에 걸쳐 자주 사용되는 데이터의 경우, 기존 방식으로 사용하면 코드가 복잡해지고 불편해진다.

컨텍스트는 리액트 컴포넌트들 사이에서 데이터를 props가 아닌 컴포넌트 트리를 통해 곧바로 컴포넌트에 전달하는 방식을 제공한다.

컨텍스트는 웹 애플리케이션의 여러 컴포넌트에서 자주 필요로 하는 사용자의 로그인 여부, 로그인 정보, UI 테마 등의 데이터를 처리할 때 무척 유용하다.

컨텍스트 API

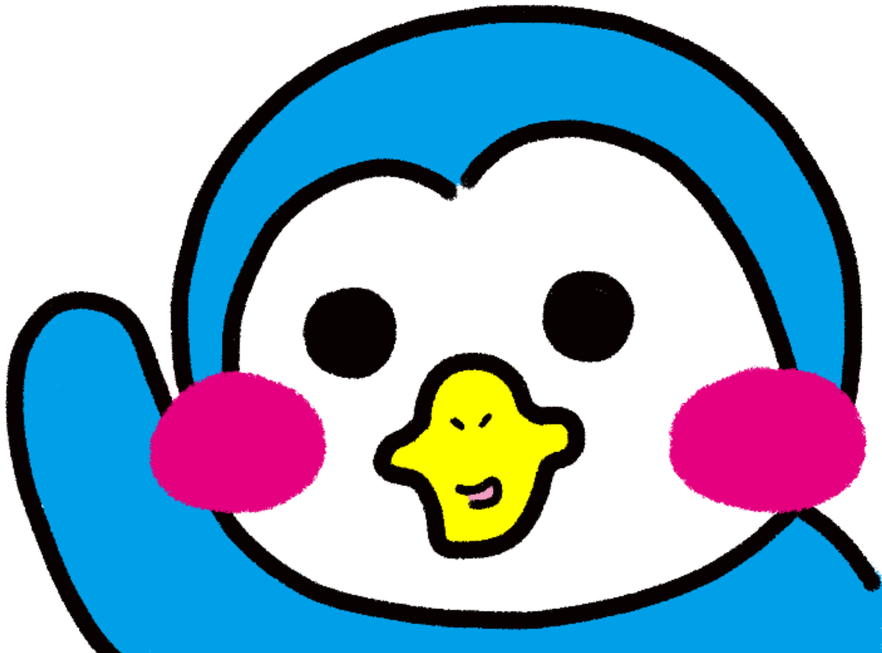
컨텍스트 활용을 위해 제공되는 대표적인 API로는 다음과 같은 것들이 존재한다.

- React.createContext : 기본값을 인수로 받아 컨텍스트 데이터를 초기화하는 함수
- Context.Provider : 하위 컴포넌트에 컨텍스트 데이터를 제공하는 래퍼 컴포넌트
- useContext : 상위 컴포넌트에서 제공된 컨텍스트 객체를 리턴하는 함수

5회차 예제, src/App06.js

프론트엔드 역량 강화 과정

비동기 통신 with axios



axios란

axios는 node.js와 브라우저를 위한 Promise 기반 HTTP 비동기 통산 라이브러리!
외부 모듈이며, 다음 명령어를 이용해 설치할 수 있다.

```
npm install axios
```

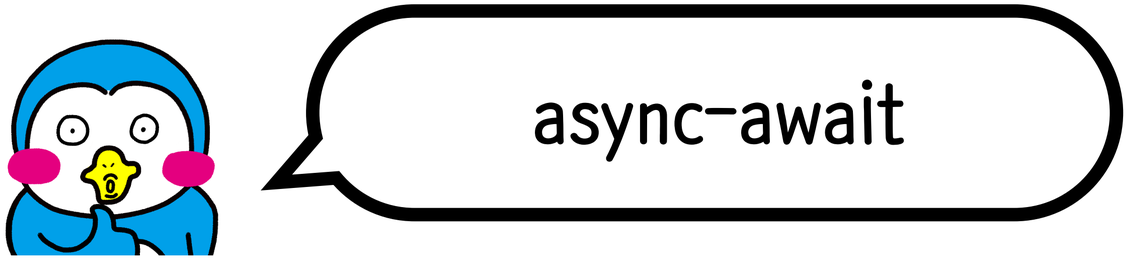
axios와 함께라면
클라이언트 사이드에서
데이터를 주고받을 수 있다!

axios 사용법

```
import axios from 'axios'

// get 요청 시
axios.get("요청주소")

// 매개변수 포함 get 요청 시
axios.get("요청주소", {
     params : {
          매개변수명: 매개변수값
     }
})
```

```
import axios from 'axios'

// post 요청 시
axios.post("요청주소", {
     매개변수명: 매개변수값,
     매개변수명: 매개변수값,
     ...
})
```

Promise를 반환한다!



axios를 이용해 Promise를 반환하는 함수를 만들 때는 async-await를 사용하기도 한다. 이때 오류 디버깅을 위해 예외 처리 구문을 추가한다.

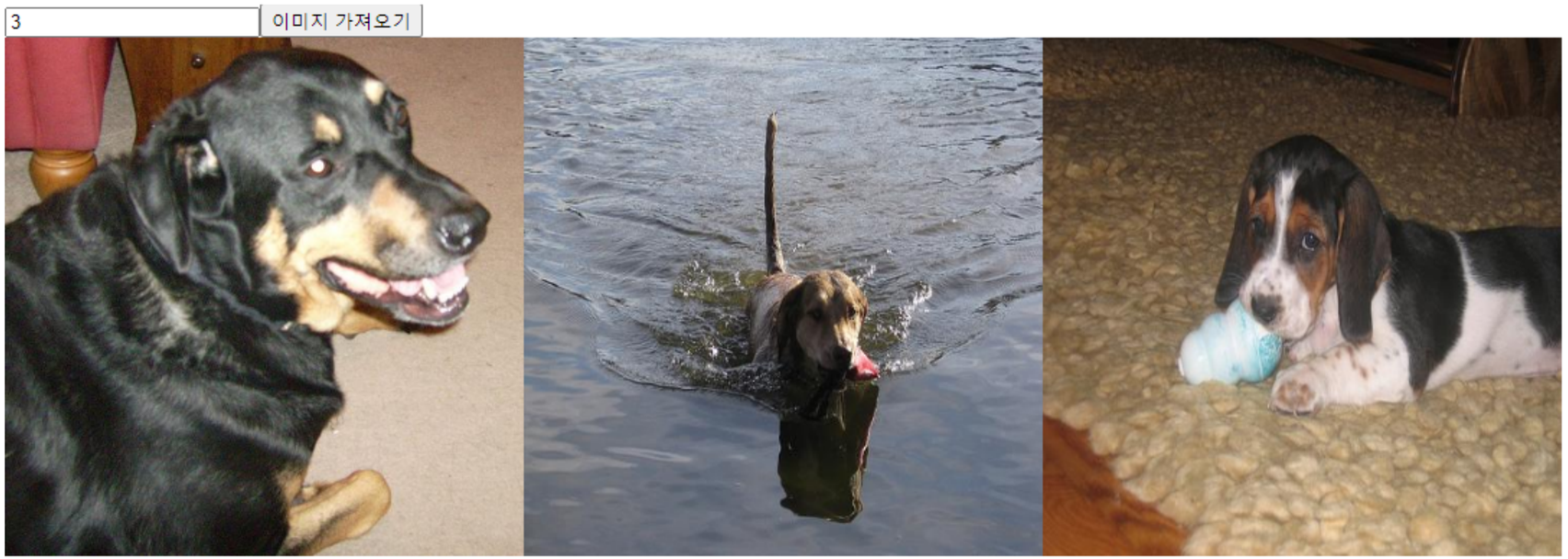
```
async function getUser() {
  try {
    const response = await axios.get('요청주소');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

5회차 예제, src/App07.js

미션 수행하기!

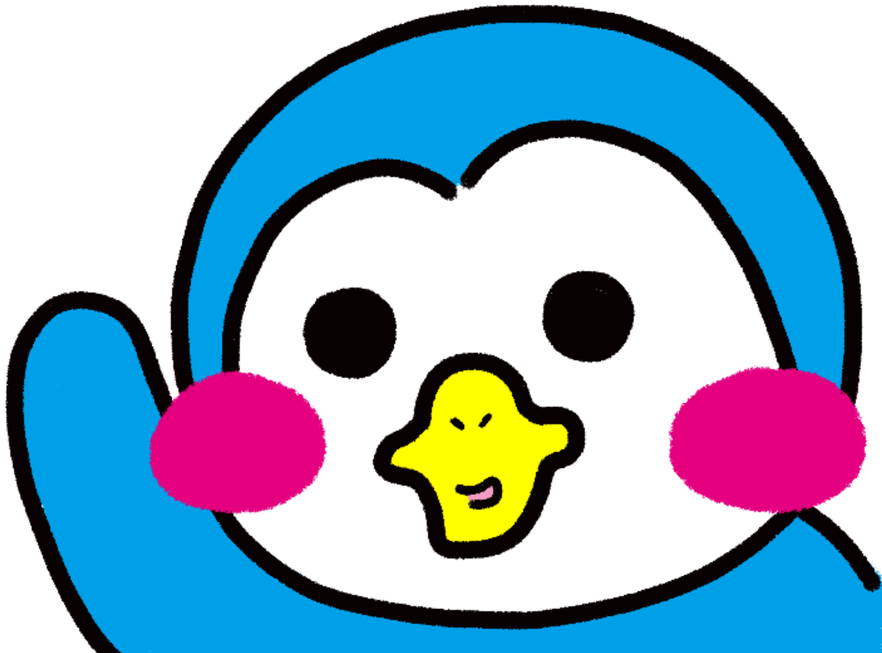
버튼을 클릭하면 강아지 이미지를 원하는 만큼 가져와 화면에 표시하는 컴포넌트를 완성해보자. src 폴더에 Task01.js 라는 파일을 만들어 처리하자.

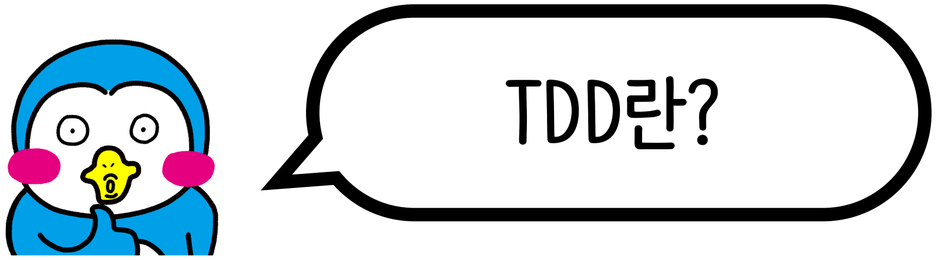
3 이미지 가져오기



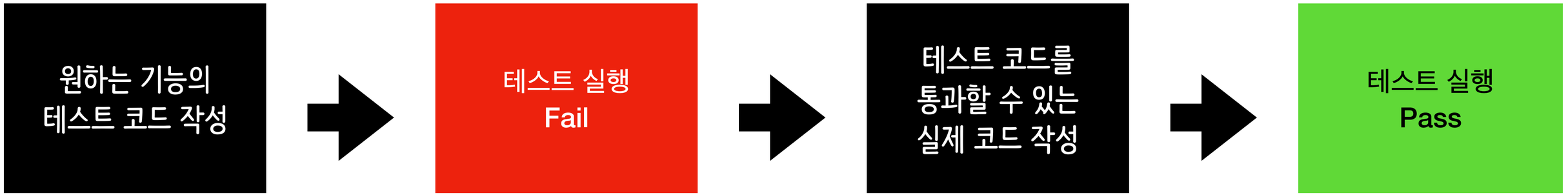
프론트엔드 역량 강화 과정

TDD





TDD(Test Driven Development)는 우리말로 '테스트 주도 개발'이라 한다.
이는 실제 코드를 작성하기 전에 테스트 코드를 먼저 작성하고, 테스트 코드를 통과할 수 있는 실제 코드를 작성하는 개발 방식을 뜻한다.



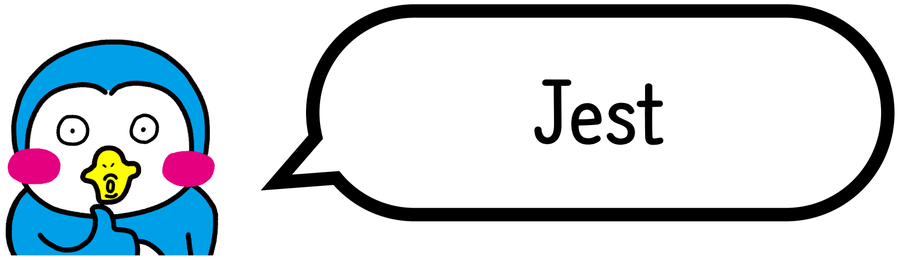
TDD를 하면 안정적으로 동작하는 코드를 작성할 수 있고, 디버깅 시간을 줄일 수 있다!

React Testing Library

DOM 노드를 테스트하고자 할 때는 DOM Testing Library 솔루션을 사용할 수 있다. 여기에 React 관련 기능을 추가하여 구축된 React Testing Library를 이용해 리액트 컴포넌트에 대한 테스트를 수행할 수 있다.

create-react-app으로 생성한 프로젝트에는 React Testing Library가 포함되어 있다. 만약 다른 방법으로 설정한 리액트 프로젝트에서 이를 사용하기 원한다면, 아래 명령어를 이용해 해당 솔루션을 설치할 수 있다.

```
npm install --save-dev @testing-library/react
```



Jest는 페이스북에서 만든 테스팅 프레임워크다.
React Testing Library와 함께 곧잘 사용되며, 아주 적은 양의 코드만으로도 단위별 테스트를 수행할 수 있는 편리한 프레임워크다.

이 또한 create-react-app 프로젝트에 기본적으로 포함되어 있다.
설치가 필요한 경우 다음 명령어를 입력하자.

```
npm install --save-dev jest
```

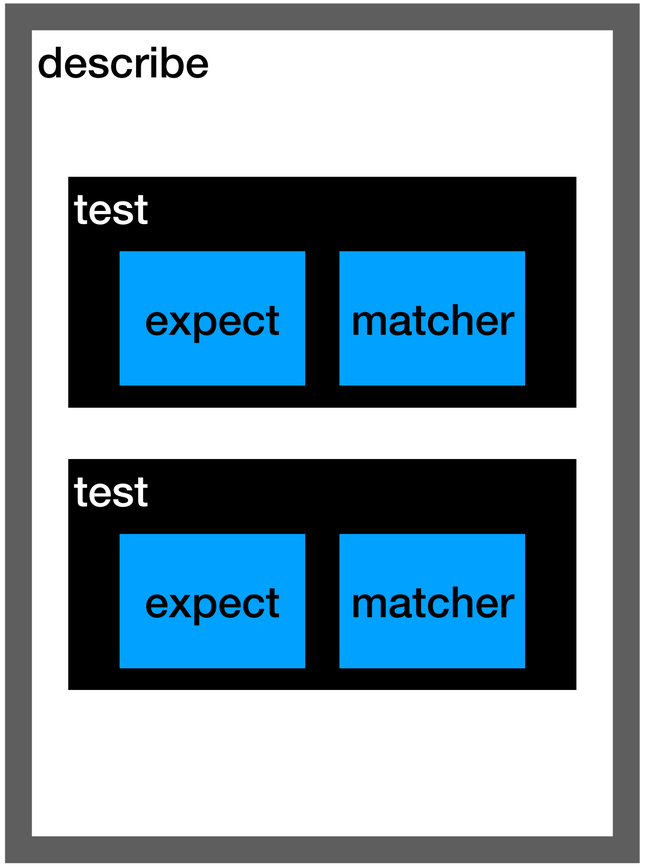
Jest 주요 API

describe : 여러 종류의 테스트를 그룹화하는 블록

it : 개별 테스트를 수행하는 블록

expect : 값을 테스트하는 메소드로, 주로 매처와 함께 사용

matcher : 값의 테스트 방법을 정의

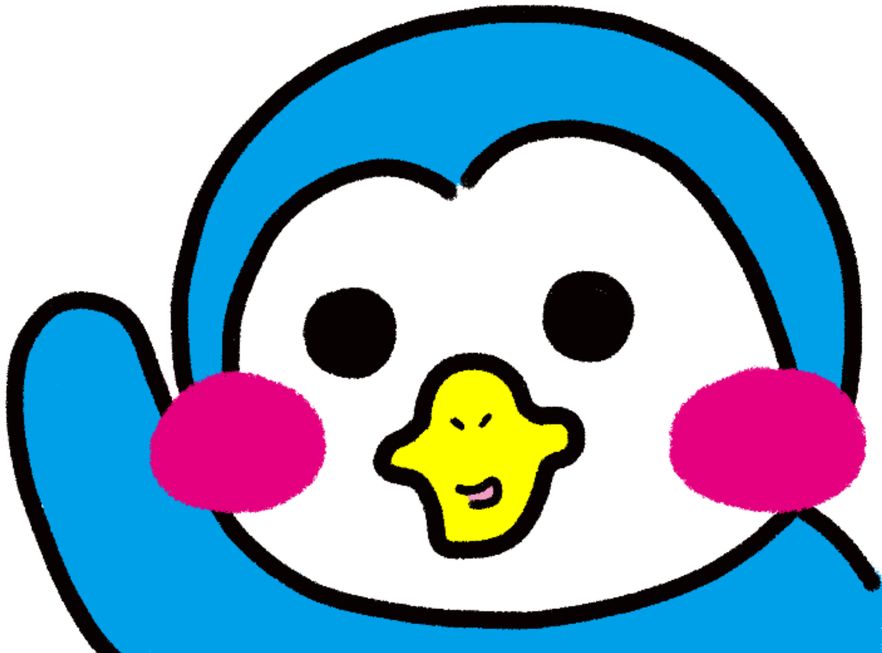


5회차 예제, src/App08.js

5회차 예제, src/App09.js

프론트엔드 역량 강화 과정

React Bootstrap



리액트 부트스트랩

리액트 부트스트랩은 스타일링된 리액트 컴포넌트를 빠르게 얻고, 기능을 커스터마이징 할 수 있도록 도와주는 라이브러리이다. 설치 명령어는 다음과 같은데, 부트스트랩과 리액트 부트스트랩을 함께 설치하여 스타일과 기능을 함께 도입하는 것이 일반적이다.

```
npm install react-bootstrap bootstrap
```

부트스트랩 스타일 적용 및 컴포넌트 선택 예

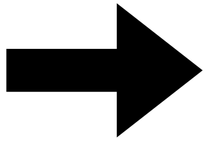
```
import ‘bootstrap/dist/css/bootstrap.min.css
import { 컴포넌트명 } from ‘react-bootstrap’
```

컴포넌트 활용하기

React Bootstrap Getting Started Components

리액트 부트스트랩 공식 웹사이트의 [components] 카테고리에서 사용할 컴포넌트를 선택하고, 개별 컴포넌트 페이지에서 제공되는 매뉴얼을 활용하면 편하고 빠르게 컴포넌트를 추가할 수 있다!

컴포넌트를 커스터마이징할 수 있도록 API 문서도 함께 제공되니 참고하자



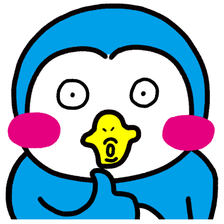
API

Card #

```
import Card from 'react-bootstrap/Card'
```

Name	Type	Default	Description
bsPrefix	string	'card'	Change the underlying component CSS base class name and modifier class names prefix. **This is an escape hatch** for working with heavily customized bootstrap css.
bg	'primary' | 'secondary' | 'success' | 'danger' | 'warning' | 'info' | 'dark' | 'light'		Sets card background
text	'primary' | 'secondary' | 'success' | 'danger' | 'warning' | 'info' | 'dark' | 'light' | 'white' | 'muted'		Sets card text color
border	'primary' | 'secondary' | 'success' | 'danger' | 'warning' | 'info' | 'dark' | 'light'		Sets card border color

5회차 예제, src/App10.js



미션 수행하기!

App10.js의 캐러셀을 활용하여 강아지 이미지 캐러셀을 만들어 보자(Task02.js).
처음 화면에는 버튼밖에 없다가, 버튼을 누르면 dog api에서 이미지를 가져와 캐러셀이 표시되도록 하자.