강의에서 사용되는 기초 React 문법

React란?

React란?

- UI(User Interface)를 효율적으로 만들기 위한 JavaScript 라이브러리
- Facebook에서 개발

React의 주요 특징

- 컴포넌트 기반: UI를 재사용 가능한 작은 조각으로 나눔
- 가상 DOM(Virtual DOM): 효율적인 업데이트 관리
- 단방향 데이터 흐름: 예측 가능한 데이터 구조

왜 React를 사용할까?

- 복잡한 UI를 쉽게 관리
- 대규모 프로젝트에서 높은 생산성과 유지보수성

DOM(Document Object Model)은 웹 페이지를 구조화하고 조작할 수 있게 해주는 모델.
정적인 HTML만으로는 할 수 없는 상황에 맞는 동작과 변화를 구현할 때 DOM이 필요.

가상 DOM(Virtual DOM)은 실제 DOM의 가벼운 복제본으로, 변경된 부분만 효율적으로 업데이트하기 위해 React에서 사용하는 메모리 기반 구조. 일반 DOM에 비해 속도가 월등하게 빠름.

```jsx
function MyButton() {
  return (
    <button>
      I'm a button
    </button>
  );
}

export default function MyApp() {
  return (
    <div>
      <h1>Welcome to my app</h1>
      <MyButton />
    </div>
  );
}
```

https://codesandbox.io/p/sandbox/hpljjw?file=%2Fsrc%2FApp.js%3A17%2C1

https://ko.react.dev/learn

DOM(Document Object Model)이란?

- DOM(Document Object Model)은 웹 페이지를 구성하는 요소들을 트리 구조로 표현

- HTML은 기본적으로 페이지의 **뼈대**를 만들고, JavaScript를 이용해 이 구조를 조작하거나 변경 가능

DOM의 주요 특징

트리 구조(Tree Structure)

- DOM은 HTML이나 XML 문서를 계층적 트리 구조로 표현.
- 문서의 각 요소(노드)는 부모, 자식, 형제 관계를 가짐.
  예: <html> → <body> → 처럼 계층적으로 연결.

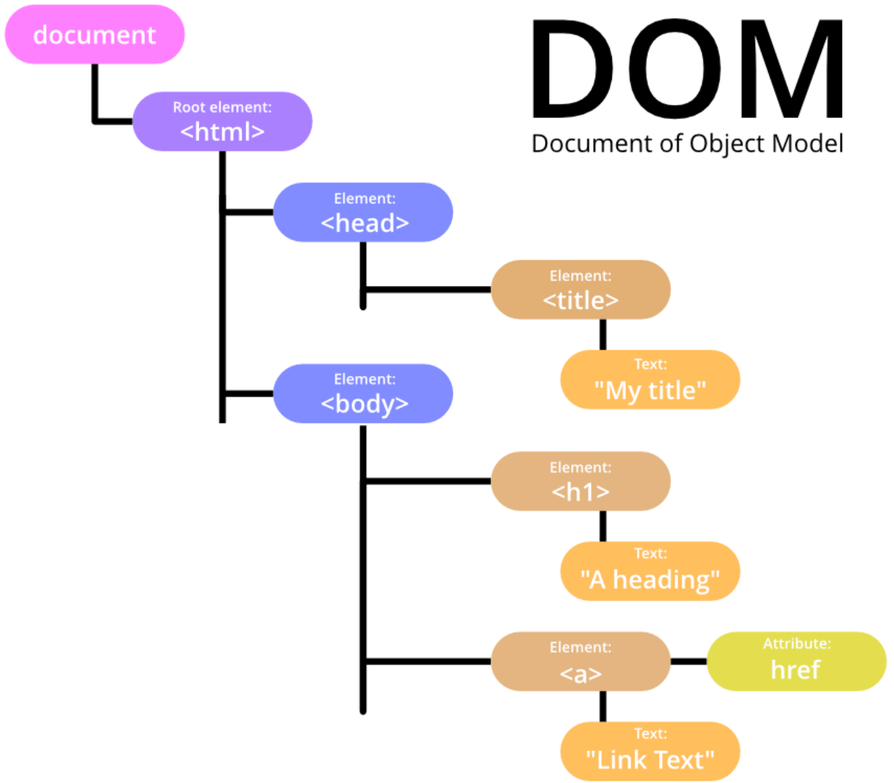
DOM을 왜 사용할까?

1. 동적 문서 조작

- DOM은 웹 페이지의 구조를 **실시간으로 추가, 삭제, 수정**

2. 실시간 반영

- DOM은 변경된 내용을 즉시 화면에 반영하여 더 나은 사용자 경험을 제공.



https://ko.react.dev/learn

React와 JavaScript의 차이

항목	JavaScript	React
정의	웹 페이지를 동적으로 만들기 위한 프로그래밍 언어	UI(화면)를 효율적으로 개발하기 위한 JavaScript 라이브러리
사용목적	모든 웹 개발 작업 가능 (로직, UI, 데이터 처리 등)	UI(화면) 개발에 특화됨
작성방식	HTML과 JavaScript를 분리하여 작성	JSX로 HTML과 JavaScript를 합쳐서 사용
DOM 처리	DOM을 직접 조작해야 함 (document.querySelector)	가상 DOM을 통해 필요한 부분만 효율적으로 업데이트
성능	DOM 조작이 많아질수록 성능 저하	가상 DOM 덕분에 많은 조작에도 성능 유지
배우기 쉬움	JavaScript는 웹 개발 입문의 기본	React는 기본 개념(JavaScript, JSX, Hooks 등) 이해 필요
컴포넌트 기반	없음	모든 화면을 컴포넌트 단위로 나누어 관리
상태 관리 방식	상태 데이터를 변수에 저장하며 직접 관리	useState, useReducer 등 React Hook을 사용해 쉽게 상태 관리

React 컴포넌트

컴포넌트란?

컴포넌트(Component)는 React에서 UI를 구성하는 **재사용 가능한 작은 코드 블록**입니다.
•하나의 컴포넌트는 화면의 한 부분(예: 버튼, 카드, 헤더 등)을 담당하며, 이를 조합해 전체 페이지를 만듭니다.

컴포넌트의 특징

1. 독립적: 컴포넌트는 독립적으로 동작하며, 각자 자신의 상태(state)와 속성(props)을 가질 수 있음.
2. 재사용 가능: 동일한 컴포넌트를 여러 곳에서 사용할 수 있어 유지보수가 쉬움.

컴포넌트의 종류

1.함수형 컴포넌트 (Functional Component):

- •JavaScript 함수처럼 작동하며, 상태 관리와 React Hook을 사용할 수 있음.
- •가장 일반적이고 권장되는 방식.

```
const Header = () => <h1>Welcome to React!</h1>;
```

2.클래스형 컴포넌트 (Class Component):

- •React 16.8 이전에 주로 사용되던 방식.
- •React Hook 도입 이후 잘 사용되지 않음.

```
class Header extends React.Component { render() { return <h1>Welcome to React!</h1>; } }
```

React Hooks란?

Hooks란?

•**React Hooks**는 React 16.8에서 도입된 기능으로, 함수형 컴포넌트에서 상태 관리와 React의 고급 기능을 사용할 수 있도록 도와주는 도구
•기존에는 상태 관리나 생명 주기 함수 등을 사용하기 위해 **클래스형 컴포넌트**가 필요했지만, Hooks를 사용하면 **함수형 컴포넌트에서도 이를 간단히 구현**

•왜 Hooks가 중요한가?

- 코드가 간결하고 직관적임.
- 함수형 컴포넌트만으로도 모든 기능을 구현 가능.
- 상태 관리와 부수 효과(예: API 호출, 이벤트 처리 등)를 더 깔끔하게 구현 가능.

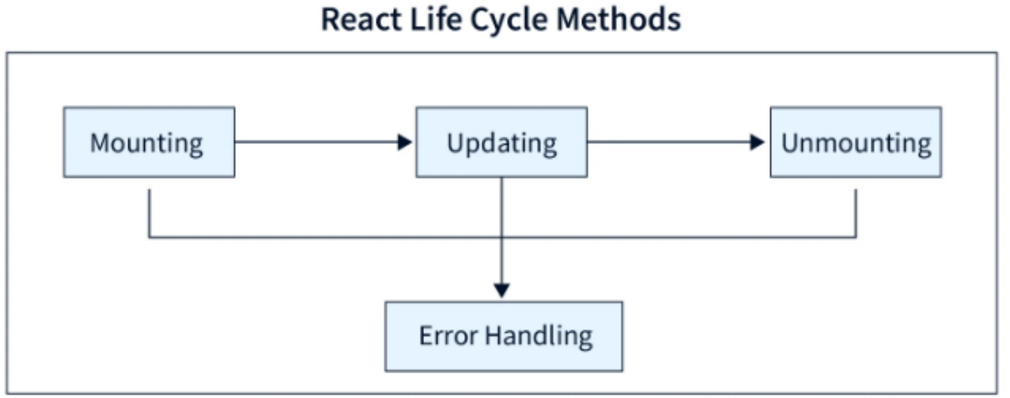
1. useState: 상태 관리
2. useEffect: 부수 효과 관리
3. useRef: DOM 참조
4. useMemo: 성능 최적화

React Hooks의 라이프사이클

React 컴포넌트는 렌더링(화면에 나타나는 것)과 관련된 생명 주기(Lifecycle)를 가집니다. 함수형 컴포넌트에서 **Hooks**를 사용해 이 라이프사이클을 제어 가능.

React 컴포넌트의 라이프사이클 단계

1.마운트(Mount): 컴포넌트가 화면에 나타날 때.
2.업데이트(Update): 상태(state)나 props가 변경되어 화면이 다시 그려질 때.
3.언마운트(Unmount): 컴포넌트가 화면에서 사라질 때.



React Hooks - useState

useState: 상태 관리

- 컴포넌트에서 상태(state)를 관리할 수 있게 해주는 Hook.
- 상태가 변경되면 React가 자동으로 화면을 다시 렌더링

const [state, setState] = useState(initialValue);
•state: 현재 상태 값.
•setState: 상태를 변경하는 함수.
•initialValue: 상태의 초기값.

```jsx
import { useState, useEffect } from "react";

export default function App() {
  const [count, setCount] = useState(0);

 return (
  <div>
   <p>Count: {count}</p>
   <button onClick={() => setCount(count + 1)}>Increment</button>
  </div>
 );
}
```

https://nextleap.app/online-compiler/reactjs-programming/vemfvpmjx

React Hooks - useEffect

useEffect: 부수 효과 관리

- 컴포넌트가 렌더링될 때 부수 효과(예: 데이터 가져오기, 구독 설정)를 처리하는 Hook.
- 클래스형 컴포넌트의 componentDidMount, componentDidUpdate, componentWillUnmount를 대체.

```
useEffect(() => {
  // 실행할 작업
  return () => {
    // 정리(cleanup) 작업 (옵션)
  };
}, [dependency]);
```

의존성 배열(dependency):
특정 상태나 값이 변경될 때만 실행.
빈 배열 []: 컴포넌트가 처음 렌더링될 때 한 번만 실행.

```jsx
import { useEffect, useState } from "react";

export default function App() {
  const [seconds, setSeconds] = useState(0); // 타이머 상태

  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds((prevSeconds) => prevSeconds + 1); // 상태 업데이트
    }, 1000);

    return () => clearInterval(interval); // 언마운트 시 정리
  }, []);

  return <p>Elapsed Time: {seconds} seconds</p>;
}
```

https://nextleap.app/online-compiler/reactjs-programming/y3m3pxab9

React Hooks - useRef

useRef: DOM 참조

- DOM 요소나 컴포넌트의 참조(reference)를 제공.
- 상태와 달리 useRef 값이 변경되어도 컴포넌트가 다시 렌더링되지 않음.
- 주로 DOM 조작이나 값 저장에 사용.

const ref = useRef(initialValue);
ref.current: 현재 참조 값.

```jsx
import { useRef } from "react";

export default function App() {
  const inputRef = useRef();

  const focusInput = () => {
    inputRef.current.focus(); // DOM 요소에 직접 접근
  };

  return (
    <div>
      <input ref={inputRef} placeholder="Click the button to focus" />
      <button onClick={focusInput}>Focus</button>
    </div>
  );
}
```

https://nextleap.app/online-compiler/reactjs-programming/nldzqts4n

React Hooks - useMemo

useMemo: 성능 최적화

- 계산 비용이 큰 작업을 메모이제이션하여 성능을 최적화.
- 컴포넌트가 렌더링될 때마다 불필요한 계산을 방지.

```
const memoizedValue = useMemo(() =>
computeExpensiveValue(input), [input]);
```

- computeExpensiveValue: 계산해야 할 함수.
- input: 값이 변경될 때만 다시 계산.

```jsx
import { useMemo, useState } from "react";

export default function App() {
  const [inputValue, setInputValue] = useState("");

  // 입력값의 길이를 메모이제이션
  const calculatedLength = useMemo(() => {
    console.log("Calculating input length...");
    return inputValue.length; // 입력값의 길이 계산
  }, [inputValue]); // inputValue가 변경될 때만 재계산

  return (
    <div>
      <input
        placeholder="Type something..."
        value={inputValue}
        onChange={(e) => setInputValue(e.target.value)} // 입력값 업데이트
      />
      <p>Input Length: {calculatedLength}</p> {/* 계산된 길이 출력 */}
    </div>
  );
}
```

https://nextleap.app/online-compiler/reactjs-programming/ax9l89gob