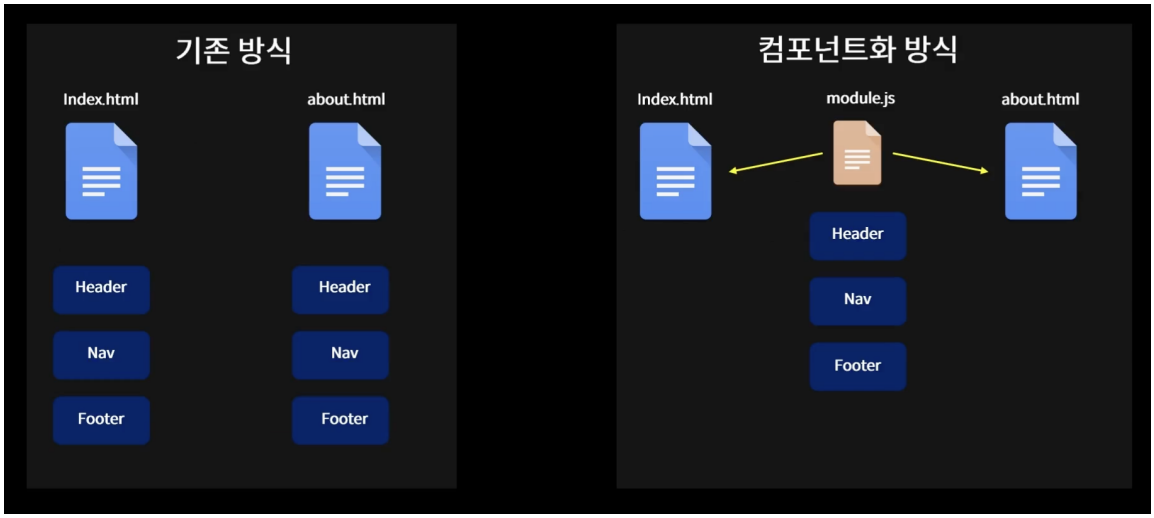
섹션 4. React.js 기초

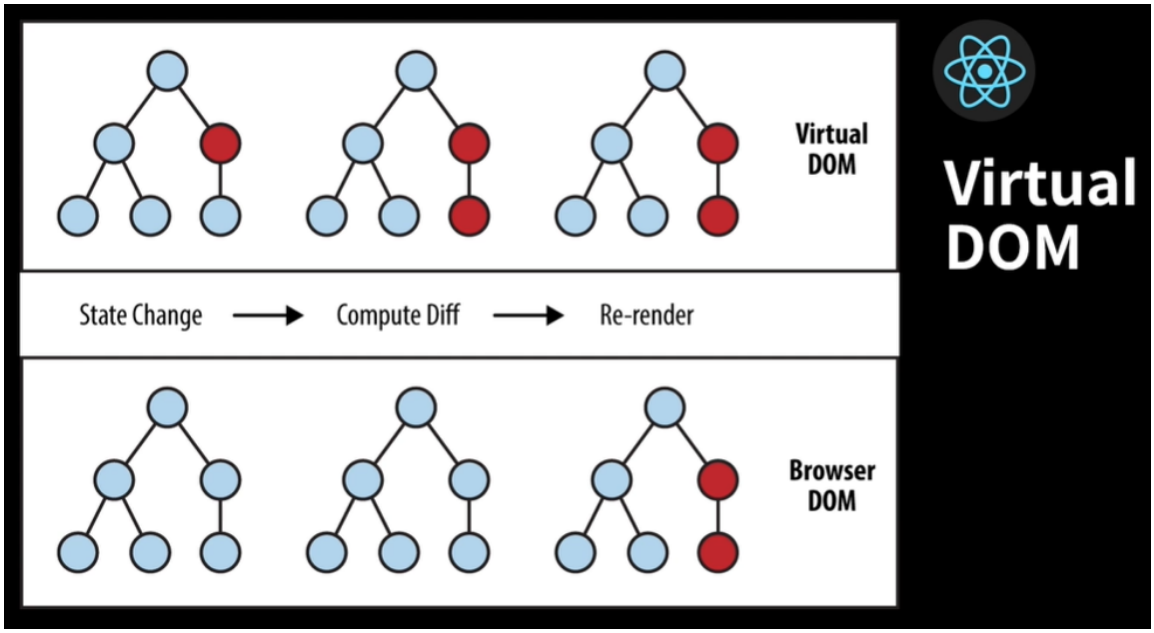
# ing	5
# goal	5
∑ 수식	✅

Why React?

1. React는 Component 기반의 UI 라이브러리



   - html 요소들을 컴포넌트화해 코드를 중복 작성 없이 재사용 가능
2. 명령형 프로그래밍 vs 선언형 프로그래밍
   - 절차를 하나하나 나열해야 하는 jQuery(명령형)에 비해 React(선언형)는 목적을 바로 말함
3. Virtual Dom
   - Dom(Document Object Model): html을 트리 형태로 변환한 객체
     - 자주 조작하면 필요 이상의 연산을 하게 되어 브라우저의 성능 저하됨
   - 가상의 DOM에 js 먼저 적용시킨 뒤 그 결과를 실제 DOM에 전달해 성능 개선



Create React App

1. 새로운 폴더에 create react app 설치
   - 터미널에서 `npx create-react-app name`
2. reactExam1 내에 있는 파일들 상위 reactExam1으로 옮기기(하위 폴더 삭제)

- 구성 요소
  - node_modules: node.js 패키지 내 구성 요소
  - public
    - favicon.ico: React 아이콘
    - index.html: index.js(메인 프로그램)에 의해 렌더링된 결과 표시
    - logo192.png, logo512.png, mainfext.json: 앱에 필요한 정보들(앱 이름, 배경색 등)
    - robots.txt: 검색 사이트에서 검색 엔진 수집 여부 지정
  - src
    - app.css: 스타일 파일
    - app.js: component 정의하는 js 파일
    - app.test.js: test용으로 사용
    - index.css: 스타일 파일
    - index.js: main.js 파일로 컴포넌트 조합 후 렌더링해 index.html에 표시

- package-lock.json: node.js 패키지의 구성 요소
- package.json: node.js 패키지의 구성 요소

JSX

- JavaScript XML: js에 xml을 추가해 확장한 문법
  - 브라우저에서 실행되기 전 babel에 의해 js 코드로 변환됨
- jsx 문법
  1. 태그 사용 시 닫는 태그 필수

```
<div></div>
<br/>
<image/>
```

  2. 최상위 태그(모든 태그를 감싸고 있는 태그)가 반드시 하나 존재해야 함

```
function component1(){
  let name = 'hi';

  return(
    <div>
      <Component0/>
      <header>
        <h2> JSX expressions must have one parent element </h2>
    </div>
  );
}
```

```
// 최상위 태그로 묶지 않을 땐 <React.Fragment> 태그 사용
import React from 'react';  // react 기능 import

function component1(){
  let name = 'hi';

  return(
    <React.Fragment>
      <Component0/>
      <header>
        <h2> JSX expressions must have one parent element </h2>
      </header>
    </React.Fragment>
  );
}
```

3. 중괄호를 사용해 js의 변수나 값 포함 가능

```
function component1(){
  let name = 'hi';
  const number = 5;
  const func1 = () => {
    return 'funcExamp';
  }

return(
  <div>
    <Component0/>
    <header>
      <h2> JSX expressions can use JS expressions </h2>
      {name}  // 변수
      {1+2}  // 값의 연산
      {'문자열'}  // 문자열
      {func1()}  // 함수
        {number}는 : {number %2 === 0 ? "짝수" : "음수"}  // 조건부 렌더링
    </header>
  </div>
);
}
```

State

- component UI의 데이터가 변경되면 자동으로 갱신되게 동적 데이터를 관리하는 object
  - component의 state가 바뀌면 해당 컴포넌트가 다시 렌더링됨

```
import React, {useState} from 'react';

const Counter = () => {
  const [count, setCount] = useState(0);  // useState 사용(매개변수: 초기값)
  const onIncrease = () => {
    setCount(count + 1);  // setCount를 통해 state를 변경해야 재렌더링됨
  }
  const onDecrease = () => {
    setCount(count - 1);
  }

  return(
    <div>
      <h1>{count}</h1>
      <button onClick=(onIncrease)> + </button>
      <button onClick=(onDecrease)> - </button>
    </div>
  )
}

export default Counter;
```

Props

- component에 props value 지정해 자식 component로 데이터 전달 가능

```
<Counter/>  // 기본
<Counter initialValue={5}/>  // 태그의 prop value 지정
```

```
// App.js 파일

import logo from './logo.svg';
import Counter from './Counter';
import './App.css';

function App(){
  const counterProps = {
    a:1, b:2, c:3, d:4, e:5,
}

return(
  <div className="App">
    <header className="App-header">
      <Counter initialValue={5} {...counterProps}/>  // spread 연산자 사용 가능
    </header>
  </div>
);
}

export default App;
// initialValue와 counterProps의 값을 전개연산자를 통해 Counter Component로 전달
```

```
// Counter.js 파일
import React, {useState} from "react"

const Counter = (props) => {
    console.log(props);
    const [count, setCount] = useState(0);

    const onIncrease = () => {
        setCount(count+1);
    }

    const onDecrease = () => {
        setCount(count-1);
    }

    return (
        <div>
            <h2>{count}</h2>
            <button onClick={onIncrease}>+</button>
            <button onClick={onDecrease}>-</button>
        </div>
```

```
    );
};

export default Counter;
// App.js에서 전달받은 initiValue와 counterProps의 값들을 매개변수로 전달받아 Counter.js에서 사용 가능
// 이때 전달받은 매개변수 = props
```

- JSX Element 요소를 전달하는 것 또한 가능

```
// Counter.js 파일
import React, {useState} from "react"
import OddEvenResult from "./OddEvenResult";
import Container from "./Container";

const Counter = () => {
    const [count, setCount] = useState(0);

    const onIncrease = () => {
        setCount(count+1);
    }

    const onDecrease = () => {
        setCount(count-1);
    }

    return (
        <Container>
            <h2>{count}</h2>
            <button onClick={onIncrease}>+</button>
            <button onClick={onDecrease}>-</button>
            <OddEvenResult count={count}/>
        </Container>
    );
};

export default Counter;
```

```
// Container.js 파일
const Container = ({children}) => {
    return (
        <div style={{margin: 20, padding: 20, border: "1px solid red"}}>
            {children}
        </div>
    );
};

export default Container;
```

- Re-render
    - component의 state가 변경될 때 해당 component는 re-render됨

- 부모 component로부터 전달받은 props가 바뀔 때마다 부모와 자식 component 둘 다 re-render됨
- 부모가 re-render되면 자식도 re-render됨