리액트 기초

**학습목표**

props 전달

이벤트 구현

리액트 상태 변수

useState 훅

**목차**

[1. props 2](#_Toc145246028)

[1) Message 컴포넌트 구현 - props 2](#_Toc145246029)

[2) PersonTable 컴포넌트 구현 - props 3](#_Toc145246030)

[3) key prop 5](#_Toc145246031)

[2. 이벤트 7](#_Toc145246032)

[1) 클릭 이벤트 7](#_Toc145246033)

[2) 변수값 변경 8](#_Toc145246034)

[3. useState 훅(hook) 10](#_Toc145246035)

[1) useState와 상태 변수 10](#_Toc145246036)

[2) React Developer Tools 11](#_Toc145246037)

[3) 상태 setter 함수 반복 호출 13](#_Toc145246038)

[4) input 구현 14](#_Toc145246039)

[5) checkbox 구현 15](#_Toc145246040)

[6) select 태그 구현 17](#_Toc145246041)

[7) radio 태그 구현 18](#_Toc145246042)

[4. 정리 20](#_Toc145246043)

[1) 요약 20](#_Toc145246044)

[2) 연습 문제 20](#_Toc145246045)

# props

## Message 컴포넌트 구현 - props

### src/App.tsx

|  |
| --- |
| import React from 'react';  import Message from './Message';  function App() {    return (      <div>        <Message value="안녕하세요" color="blue" />      </div>    );  }  export default App; |

<Message value="안녕하세요" color="blue" />

위 코드는 Message 함수형 컴포넌트를 호출한다.

이때 value="안녕하세요" color="blue" props 값들을 Message 함수에 전달한다.

이렇게 전달되는 props 값들은 객체로 묶여서 Message 함수에 전달된다.

즉 { value: "안녕하세요", color: "blue" } 형태의 객체가 Message 함수에 전달된다.

### src/Message.tsx

|  |
| --- |
| import React from 'react'  export type MessageProps = {      value: string,      color: string  }  function Message(props: MessageProps) {    return (      <div style={{color: props.color}}>{ props.value }</div>    )  }  export default Message; |

function Message(props: MessageProps) {

함수형 컴포넌트에 전달된 props 값들이 Message 함수의 파라미터 변수로 전달된다.

export type MessageProps = {

    value: string,

    color: string

}

이 부분은 Message 컴포넌트의 props 값 객체의 타입 선언이다.

props 값이 위 선언과 일치하지 않으면 컴파일 에러가 발생한다.

Graphical user interface, text, application, chat or text message

Description automatically generated

## PersonTable 컴포넌트 구현 - props

### src/PersonTable.css 생성

|  |
| --- |
| table.PersonTable { border-collapse: collapse; margin: 5px; }  table.PersonTable td { border: 1px solid gray; padding: 4px; } |

### src/PersonTable.tsx 생성

|  |
| --- |
| import React from 'react';  import './PersonTable.css';  export type Person = {    name: string,    age: number  };  export type PersonTableProps = {      persons: Person[]  }  function PersonTable(props: PersonTableProps) {    let trlist = props.persons.map(person =>          <tr><td>{ person.name }</td><td>{ person.age }</td></tr>);    return (      <table className="PersonTable">        <tr><td>이름</td><td>나이</td></tr>        { trlist }      </table>    );  }  export default PersonTable; |

### src/App.tsx

|  |
| --- |
| import React from 'react';  import type { Person } from './PersonTable';  import PersonTable from './PersonTable';  function App() {    let persons1 : Person[] = [{ name: '홍길동', age: 16 },                               { name: '임꺽정', age: 19 },                               { name: '전우치', age: 20 }];    let persons2 = persons1.slice(0);    persons2.reverse();    return (      <div>        <PersonTable persons={ persons1 } />        <hr />        <PersonTable persons={ persons2 }  />      </div>    );  }  export default App; |

import type { Person } from './PersonTable';

PersonTable.tsx 파일에서 export type된 Person 타입을 import 한다.

import PersonTable from './PersonTable';

PersonTable.tsx 파일에서 export default된 PeronsTable 함수를 import 한다.

let persons2 = persons1.slice(0);

persons2.reverse();

person1 배열을 복제하여 person2 배열을 만들고

person2 배열의 순서를 뒤집는다.

<PersonTable persons={ persons1 } />

PersonTable 컴포넌트를 호출하면서 persons1 배열을 props로 전달한다.

persons1 부분은 자바스크립트 표현식이므로 { } 괄호로 묶어야 한다.

Text

Description automatically generated

## key prop

F12 키를 눌러서 웹브라우저 개발자창을 열고, Console 탭을 보자.

Graphical user interface, text, application, email

Description automatically generated

Warning: Each child in a list should have a unique "key" prop

태그 목록의 각각의 항목에는 "key" prop가 있어야 한다는 경고

Warning: validateDOMNesting(...): <tr> cannot appear as a child of <table>. Add a <tbody>, <thead> or <tfoot> to your code to match the DOM tree generated by the browser.

<table> 태그 바로 아래에 <tr> 태그가 오면 안되고, <thead> <tbody> 태그가 필요하다는 경고

### "key" prop가 있어야 하는 이유

<tr> 태그나 어떤 태그가 반복된 목록의 경우, 그 목록 태그에 "key" prop가 있어야 한다.

그리고 "key" prop의 값은 DB 테이블의 기본키(primary key) 같은 유일한 값이어야 한다.

리액트는 데이터가 수정 삭제될 때 마다,

전체 코드를 다시 실행해서 화면 Virtual DOM 객체들을 전부 다시 생성하고,

직전에 생성했던 Virtual DOM 객체들과 비교하여 달라진 부분을 찾는다.

이렇게 달라진 부분을 찾을 때, 목록을 구성하는 태그에 "key" prop이 있으면

좀 더 정확하고 빠르게 달라진 부분을 찾을 수 있다.

### 주의

배열의 index를 key 값으로 사용하는 예제 코드를 가끔 볼 수 있는데, 잘못된 구현이다.

기본키(primary key) 값 같은 것을 사용해야 한다.

### src/PersonTable.tsx 수정

|  |
| --- |
| import React from 'react';  import './PersonTable.css';  export type Person = {    id: number,    name: string,    age: number  };  export type PersonTableProps = {      persons: Person[]  }  function PersonTable(props: PersonTableProps) {    let trlist = props.persons.map(person =>          <tr key={ person.id }><td>{ person.name }</td><td>{ person.age }</td></tr>);    return (      <table className="PersonTable">        <tbody>          <tr key={0}><td>이름</td><td>나이</td></tr>          { trlist }        </tbody>      </table>    );  }  export default PersonTable; |

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import type { Person } from './PersonTable';  import PersonTable from './PersonTable';  function App() {    let persons1 : Person[] = [{ id: 31, name: '홍길동', age: 16 },                               { id: 32, name: '임꺽정', age: 19 },                               { id: 33, name: '전우치', age: 20 }];    let persons2 = persons1.slice(0);    persons2.reverse();    return (      <div>        <PersonTable persons={ persons1 } />        <hr />        <PersonTable persons={ persons2 }  />      </div>    );  }  export default App; |

# 이벤트

## 클릭 이벤트

### src/App.css 수정

|  |
| --- |
| button { font-size: 12pt; padding: 4px 20px; margin: 8px; } |

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    const onClickHandler = () => alert('버튼1 클릭');    return (      <div>        <button onClick={onClickHandler}>버튼1</button>        <button onClick={() => alert('버튼2 클릭')}>버튼2</button>      </div>    );  }  export default App; |

### 이벤트 이름

HTML DOM 객체 이벤트 이름과 Virtual DOM 이벤트 이름이 약간 다르다.

예를 들어 클릭 이벤트의 경우 HTML에서는 onclick이지만,

Virtual DOM에서는 onClick이다.

Virtual DOM 이벤트 이름은 카멜 케이스(camel case)로 표기해야 한다.

### 이벤트 구현 방법

onClick 이벤트의 값은 자바스크립트 함수이어야 한다.

자바스크립트 함수는 당연히 자바스크립트 코드이므로 { } 괄호로 감싸야 한다.

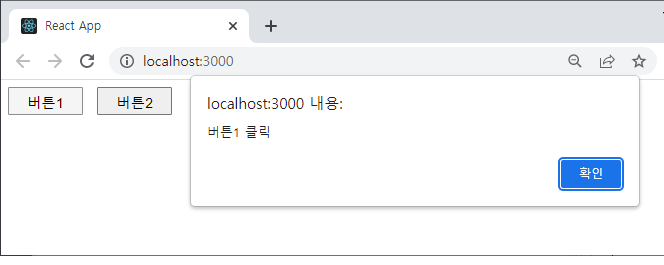
const onClickHandler = () => alert('버튼1 클릭');

위와 같이 함수를 미리 구현한 후, 다음과 같이 지정해줄 수 있다

<button onClick={onClickHandler}>버튼1</button>

아니면 다음과 같이 바로 함수를 구현할 수도 있다.

<button onClick={() => alert('버튼2 클릭')}>버튼2</button>



## 변수값 변경

### src/App.css 수정

|  |
| --- |
| button { font-size: 12pt; padding: 4px 20px; margin: 8px; }  p { font-size: 20pt; padding: 0px 20px; } |

### src/App.tsx 수정

|  |
| --- |
| import React from 'react';  import './App.css';  function App() {    let count = 0;    const increase = () => { ++count; console.log(count); };    const decrease = () => { --count; console.log(count); };    return (      <div>        <p>{ count }</p>        <button onClick={increase}>증가</button>        <button onClick={decrease}>감소</button>      </div>    );  }  export default App; |

Graphical user interface, text, application, email

Description automatically generated

증가 버튼을 클릭하면 increase 함수가 호출되어 ++count 된다.

감소 버튼을 클릭하면 decrease 함수가 호출되어 --count 된다.

그리고 <p>{ count }</p> 부분에 count 변수값이 출력되어야 한다.

그런데 버튼을 클릭해되 count 변수값 출력에 변화가 없다.

Graphical user interface, text, application, email

Description automatically generated

개발자창의 console 탭을 열어보면, 버튼이 클릭될 때 마다 count 변수값이 변경되고 있음을 알 수 있다.

count 변수 값은 변경되었지만, 화면이 다시 그려지지 않은 것이다. (렌더링 되지 않는 것)

변수값이 변경되다고 해서, 리액트 앱 화면이 다시 렌더링 되지 않는다.

# useState 훅(hook)

## useState와 상태 변수

평범한 자바스크립트 변수의 값이 변경될 때, 화면은 다시 렌더링 되지 않는다.

어떤 값이 변경될 때 마다 화면이 다시 렌더링 되어야 한다면,

그 값은 평범한 자바크립트 변수 대신 리액트 상태 변수에 저장되어야 한다.

즉 리액트 상태 변수의 값이 변경될 때는 화면이 다시 렌더링 된다.

### src/App.tsx 수정

|  |
| --- |
| import React, { useState } from 'react';  import './App.css';  function App() {    const [count, setCount] = useState(0);    const increase = () => { setCount(count + 1); };    const decrease = () => { setCount(count - 1); };    return (      <div>        <p>{ count }</p>        <button onClick={increase}>증가</button>        <button onClick={decrease}>감소</button>      </div>    );  }  export default App; |

### useState 훅

리액트에는 이름이 use로 시작하는 특별한 함수들이 제공된다.

이 함수들을 함수라고 부르지 않고 훅(hook)이라고 부른다.

useState 훅을 이용해서 리액트 상태 변수를 만들어야 한다.

import React, { useState } from 'react';

useState 훅 import

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

리액트 상태 변수 만드는 코드.

이렇게 만들어진 상태 변수는 리액트 내부에 만들어진다.

0 부분은 그 상태 변수의 최초 값이다.

그 상태 변수의 현재 값이 리턴되어 count 상수에 대입된다.

그 상태 변수 값을 변경하려면, setCount 함수를 호출해야 한다.

위 코드는 구조 분해 할당 문법이다.

화면이 다시 렌더링 될 때마다, App 함수가 다시 호출되고

아래 코드도 다시 다시 실행된다.

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

위 코드가 실행될 때 마다 상태 변수가 만들어지는 것은 아니고,

위 코드가 최초 실행될 때에만 만들어진다.

setCount 함수를 호출하면, 리액트 내부에 만들어진 상태 변수 값이 변경되고,

다음 렌더링 때 위 코드가 다시 호출될 때 그 값이 리턴되어 count 상수에 대입된다.

즉 setCount 함수를 호출하자마자 count 상수값이 변경되는 것이 아니다.

Graphical user interface, text, application, email

Description automatically generated

증가 감소 버튼을 클릭할 때 마다 화면이 렌더링 된다.

## React Developer Tools

chrome에서 React Developer Tools 확장을 검색해서 설치하자.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

chrome 개발자창에서 Components 탭을 열면,

리액트 상태 변수들을 볼 수 있다.

|  |
| --- |
| 참고:  const [count, setCount] = useState(0);  위 코드로 count 리액트 상태 변수가 만들어진다고 설명했지만, 편의상 그렇게 설명한 것이고 사실은 아니다.  useState 훅을 호출하면서 상태 변수 이름이 count 이라고 알려주지 않았다.  단지 초기값 0만 파라미터로 전달했다.  리액트 상태 변수는, 어떤 컴포넌트에서 몇 번째로 만든 것인가?로 구별된다.  예를 들어 App 함수에서 useState 훅을 3번 호출한다면  1번 상태 변수,2번 상태 변수,3번 상태 변수가 만들어진다.  위 화면에서, App 컴포넌트에서 만든 1번 상태 변수의 현재 값이 5 라는 것을 확인할 수 있다.  위 화면에서 상태 값을 수정하면 화면이 다시 렌더링 된다. |

## 상태 setter 함수 반복 호출

### src/App.tsx 수정

|  |
| --- |
| import React, { useState } from 'react';  import './App.css';  function App() {    const [count, setCount] = useState(0);    const increase = () => {      setCount(count + 1);      setCount(count + 1);    };    const decrease = () => {      setCount(count - 1);      setCount(count - 1);    };    return (      <div>        <p>{ count }</p>        <button onClick={increase}>증가</button>        <button onClick={decrease}>감소</button>      </div>    );  }  export default App; |

코드를 위와 같이 수정하면 실행하면 어떻게 될까?

setCount(count + 1);

버튼을 클릭할 때 마다 위 코드가 두 번씩 호출되니 count 값이 2씩 증가할까?

그렇지 않다.

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

화면이 렌더링 될 때 마다 위 코드도 다시 호출된다.

현재 상태 변수값이 리턴되어 count 상수에 대입된다.

만약 그 값이 3 이라면,

    setCount(count + 1);

    setCount(count + 1);

위 코드는 다음과 같다.

    setCount(3 + 1);

    setCount(3 + 1);

setCount 함수는 리액트 내부의 상태 변수 값을 변경하는 것이지,

count 상수 값을 변경하는 것이 아니다.

setCount 함수 호출로 변경된 상태 변수 값은

다음 렌더링 때 아래 코드가 다시 호출될 때 그 값이 리턴되어 count 상수에 대입된다.

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

## input 구현

### src/App.css 수정

|  |
| --- |
| p { font-size: 20pt; padding: 0px 20px; }  input { font-size: 12pt; padding: 5px; margin: 20px; } |

### src/App.tsx 구현

|  |
| --- |
| import React, { useState } from 'react';  import './App.css';  function App() {    const [message, setMessage] = useState('hello');    return (      <div>        <p>{ message }</p>        <input type="text" onChange={(e) => setMessage(e.target.value)} value={message} />      </div>    );  }  export default App; |

리액트에서 input 태그는 위와 같이 구현해야 한다.

const [message, setMessage] = useState('hello');

<input type="text" onChange={(e) => setMessage(e.target.value)} value={message} />

렌더링 될 때, message 상태 값이 input 태그에 채워진다.

input 태그에 값이 입력되면, setMessage 함수가 호출되어 입력된 값이 message 상태에 채워진다.

즉 message 상태 값과 input 태그가 양방향 연결된다.

사용자가 input 태그에 문자를 하나 입력할 때 마다,

onChange 이벤트가 발생하고

setMessage 함수가 호출되어 입력된 값이 message 상태 값에 채워진다.

상태 값이 변경되었으므로, 화면이 다시 렌더링 된다.

즉 사용자가 input 태그에 문자를 하나 입력할 때 마다, 위 App 컴포넌트가 다시 렌더링된다.

꽤나 비효율적인 방식인 것 같지만, 요즘 CPU가 워낙 빨라서 별로 문제되지 않는다.

## checkbox 구현

### src/App.css

|  |
| --- |
| div#app { margin: 20px auto; padding: 20px; width: 300px;    border: 1px solid gray; box-shadow: 5px 5px 5px #aaa; }  p { font-size: 20pt; color: blue; }  input { width: 30px; height: 30px;  } |

### src/App.tsx

|  |
| --- |
| import React, { useState } from 'react';  import './App.css';  function App() {    const [value, setValue] = useState(true);    return (      <div id="app">        <p>{ value ? "true" : "false" }</p>        <hr />        <input type="checkbox" checked={value}               onChange={(e) => setValue(e.target.checked)} />      </div>    );  }  export default App; |

Graphical user interface, text, application

Description automatically generated

const [value, setValue] = useState(true);

<input type="checkbox" checked={value}

       onChange={(e) => setValue(e.target.checked)} />

렌더링 될 때, value 상태 값 true/false에 따라 체크박스의 체크 상태가 설정된다.

체크박스 체크 상태가 변경되면, setValue 함수가 호출되어

체크 상태 true/false가 value 상태에 채워진다.

즉 value 상태 값과 체크박스가 양방향 연결된다.

### src/App.css 수정

|  |
| --- |
| div#app { margin: 20px auto; padding: 20px; width: 300px;      border: 1px solid gray; box-shadow: 5px 5px 5px #aaa; }  p { font-size: 20pt; color: blue; }  input { width: 30px; height: 30px;  }  div#hello { margin-top: 20px; padding: 20px; background-color: lightGreen;  border: 1px solid gray; text-align: center; font-size: 20pt; } |

### src/App.tsx 수정

|  |
| --- |
| import React, { useState } from 'react';  import './App.css';  function App() {    const [value, setValue] = useState(true);    const helloTag = <div id="hello">Hello React.js</div>;    return (      <div id="app">        <p>{ value ? "true" : "false" }</p>        <hr />        <input type="checkbox" checked={value}               onChange={(e) => setValue(e.target.checked)} />        { value ? helloTag : null }      </div>    );  }  export default App; |

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

const helloTag = <div id="hello">Hello React.js</div>;

helloTag 상수 변수에 대입된 것은 div 태그 Virtual DOM 자바스크립트 객체이다.

App 함수는 Virtual DOM 객체들을 생성하여 리턴한다.

리턴되는 Virtual DOM 객체들은 트리(tree) 구조를 이루고 있다.

트리의 루트는 <div id="app"> 태그 객체이다.

{ value ? helloTag : null }

value 값이 true이면 위 표현식의 값은 helloTag 상수 변수에 대입된 Virtual DOM 객체이고

이 Virtual DOM 객체가 트리의 이 위치에 삽입된다.

false이면 위 표현식의 값은 null 이고, 트리에 아무것도 삽입되지 않는다.

## select 태그 구현

### src/App.css

|  |
| --- |
| div#app { margin: 20px auto; padding: 20px; width: 300px;    border: 1px solid gray; box-shadow: 5px 5px 5px #aaa; }  select { padding: 5px; font-size: 15pt; width: 150px; margin: 10px; }  .yellow { background-color: #ffa; }  .green { background-color: #bfb; }  .blue { background-color: #bbf; }  .red { background-color: #fbb; } |

### src/App.tsx

|  |
| --- |
| import React, { useState } from 'react';  import './App.css';  function App() {    const [color, setColor] = useState('green');    return (      <div id="app" className={color}>        <h1>select</h1>        <select onChange={(e) => setColor(e.target.value)} value={color}>          <option>yellow</option>          <option>green</option>          <option>blue</option>          <option>red</option>        </select>      </div>    );  }  export default App; |

Graphical user interface, application

Description automatically generated

const [color, setColor] = useState('green');

<select onChange={(e) => setColor(e.target.value)} value={color}>

렌더링 될 때, color 상태 값에 해당하는 option 태그가 선택된다.

선택이 변경되면, setColor 함수가 호출되어 선택된 option 태그의 값이 color 상태에 채워진다.

즉 color 상태 값과 select 태그가 양방향 연결된다.

<div id="app" className={color}>

예를 들어 color 상태 변수 값이 yellow로 바뀌면

className 값은 yellow가 되고

App.css 파일에 정의된 .yellow CSS 스타일이 적용된다.

## radio 태그 구현

### src/App.css 수정

|  |
| --- |
| div#app { margin: 20px auto; padding: 20px; width: 300px;      border: 1px solid gray; box-shadow: 5px 5px 5px #aaa; }  p { font-size: 18pt; color: blue; }  input { width: 20px; height: 20px;  }  label { margin-right: 30px; font-size: 14pt; } |

### src/App.tsx 수정

|  |
| --- |
| import React, { useState } from 'react';  import './App.css';  function App() {    const [gender, setGender] = useState('여자');    return (      <div id="app">        <p>{ gender }</p>        <hr />        <label>          <input type="radio" name="gender"                 onChange={(e) => setGender('남자')} checked={ gender === '남자' } />          <span>남자</span>        </label>        <label>          <input type="radio" name="gender"                 onChange={(e) => setGender('여자')} checked={ gender === '여자' } />          <span>여자</span>        </label>      </div>    );  }  export default App; |

Graphical user interface, text, application

Description automatically generated

const [gender, setGender] = useState('여자');

<input type="radio" name="gender"

       onChange={(e) => setGender('남자')} checked={ gender === '남자' } />

<input type="radio" name="gender"

       onChange={(e) => setGender('여자')} checked={ gender === '여자' } />

렌더링 될 때,

checked={ gender === '남자' } 조건식이 true 이면 첫번째 radio 버튼이 체크되고,

checked={ gender === '여자' } 조건식이 true 이면 두번째 radio 버튼이 체크된다.

radio 버튼의 선택이 변경되면,

onChange={(e) => setGender('남자')} 부분이나

onChange={(e) => setGender('여자')} 부분 중 하나가 실행되어

gender 상태에 '남자', '여자' 둘 중의 하나가 채워진다.

즉 gender 상태 값과 radio 버튼이 양방향 연결된다.

# 정리

## 요약

### props

<Message value="안녕하세요" color="blue" />

위 코드는 Message 함수형 컴포넌트를 호출한다.

이때 { value: "안녕하세요", color: "blue" } 형태의 props 값 객체가 Message 함수에 전달된다.

Message 함수 컴포넌트는 props 값 객체를 파라미터로 받는다.

export type MessageProps = {

    value: string,

    color: string

}

function Message(props: MessageProps) {

  return ...

}

### "key" prop

<tr> 태그나 어떤 태그가 반복된 목록의 경우, 그 목록 태그에 "key" prop가 있어야 한다.

그리고 "key" prop의 값은 DB 테이블의 기본키(primary key) 같은 유일한 값이어야 한다.

### 이벤트 구현

onClick 이벤트의 값은 자바스크립트 함수이어야 한다.

자바스크립트 함수는 당연히 자바스크립트 코드이므로 { } 괄호로 감싸야 한다.

const onClickHandler = () => alert('버튼1 클릭');

<button onClick={onClickHandler}>버튼1</button>

<button onClick={() => alert('버튼2 클릭')}>버튼2</button>

### useState 훅

평범한 자바스크립트 변수의 값이 변경될 때, 화면은 다시 렌더링 되지 않는다.

useState 훅을 이용해서 리액트 상태 변수를 만들어야 한다.

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

## 연습 문제

학생 이름, 학번, 학과를 입력하는 입력 폼을 구현하라.

학과 입력은 select 태그로 구현하라.