React.js

이벤트 처리, 조건부 렌더링, 리스트와 키, Form

first coding

React.js

1. 이벤트 처리
2. 조건부 렌더링
3. 리스트와 키
4. Form

first
coding

이벤트 처리

이벤트 처리하기

- **이벤트 처리하기**
  - DOM 에서 이벤트처리

```
<button onclick="activate()">
    activate
</button>
```

  - 리액트 코드

```
<button onClick={activate}>
    activate
</button>
```

    - 이벤트의 이름인 onclick이 onClick으로 카멜 표기법이 적용
    - DOM에서는 이벤트를 처리할 함수를 문자열로 전달하지만 리액트에서는 함수 그대로 전달

이벤트 처리하기

- 이벤트 처리하기
  - 이벤트가 발생했을 때 해당 이벤트를 처리하는 함수가 있는데 이것을 이벤트 핸들러 또는 이벤트 리스너라고 함.
  - 이벤트 핸들러 추가방법

```
class Toggle extends React.Component {

    constructor(props) {
        super(props);
        this.state = { isToggleOn: true };
        // callback에서 'this' 를 사용하기 위해서는 바인딩을 필수적으로 해줘야 한다.
        this.handleClick = this.handleClick.bind(this);
    }

    handleClick() {
        this.setState(prevState =>({
            isToggleOn: !prevState.isToggleOn
        }));
    }

    render() {
        return (
            <button onClick={this.handleClick}>
                {this.state.isToggleOn ? '켜짐' : '꺼짐'}
            </button>
        );
    }
}
```

- **이벤트 처리하기**
  - 먼저 handleClick() 함수의 정의 부분은 일반적인 함수를 정의하는 것과 동일하게 괄호와 중괄호를 사용해서 클래스의 함수로 정의하고 함수를 constructor()에서 bind()를 이용하여 this.handleClick에 대입해 준다.
  - 자바스크립트에서는 기본적으로 클래스 함수들이 바운드되지 않기 때문에 JSX에서 this 의 의미에 대해 유의해야 한다.
    - bind를 하지 않으면 this.handleClick 은 글로벌 스코프global scope에서 호출되는데 글로벌 스코프에서 this.handleClick 은 undefined 이므로 사용할 수가 없다.

      ➔ 일반적으로 함수의 이름 뒤에 괄호 없이 사용하려면 무조건해당 함수를 bind 해줘야 한다.

이벤트 처리하기

- 이벤트 처리하기
  - 함수 컴포넌트로 변경
    - 함수 컴포넌트에서는 this를 사용하지 않고 아래 코드처럼 onClick에 곧바로 handleClick을 넘기면 됨.

```
function Toggle(props) {

    const [isToggleOn, setIsToggleOn] = useState(true)

    // 방법1. 함수안에 함수로 정의
    function handleClick1() {
        setIsToggleOn(isToggleOn => !isToggleOn)
    }

    // 방법1. 함수안에 함수로 정의
    const handleClick2 = () => {
        setIsToggleOn(isToggleOn => !isToggleOn)
    }

    return (
        <button onClick={handleClick1}>
            {this.state.isToggleOn ? '켜짐' : '꺼짐'}
        </button>
    );
}
```

Arguments 전달하기

- Arguments 전달하기
  - 이벤트 핸들러에 Arguments(매개변수)를 전달하는 방법.

① `<button onClick={(event) => this.deleteItem(id, event)}>삭제하기</button>`

② `<button onClick={this.deleteItem.bind(this, id)>삭제하기</button>`

- 하나는 arrow function을 사용했고 다른 하나는 Function.prototype.bind를 사용
- event라는 매개변수는 리액트의 이벤트 객체를 의미
- 두 방법 모두 첫 번째 매개변수는 id이고 두 번째 매개변수로 event가 전달.

- arrow function을 사용한 방법은 명시적으로 event를 두 번째 매개변수로 넣어 주었고
- Function.prototype.bind를 사용한 방법은 this가 event로 전달 되고 id 가 두 번째 매개변수로 전달

Arguments 전달하기

- Arguments 전달하기
  - 함수 컴포넌트에서의 핸들러에 매개변수 전달

```
function Mybutton(props) {

    const handleDelete = (id, event) => {
        console.log(id, event.target);
    };

    return (
        <button onClick={ (event) => handleDelete(1, event) }>
            삭제하기
        </button>
    );

}
```

클릭 이벤트 처리하기

- ConfirmButton.jsx

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

function ConfirmButton(props) {

    const [isConfirmed, setIsConfirmed] = useState(false);

    const handleConfirm = () => {
        setIsConfirmed((prevIsConfirmed) => !prevIsConfirmed);
    };

    return (
        <button onClick={handleConfirm} disabled={isConfirmed}>
            {isConfirmed ? "확인됨" : "확인하기"}
        </button>
    );

}

export default ConfirmButton;
```

클릭 이벤트 처리하기

- Index.jsx

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <ConfirmButton  />
);
```

first
coding

조건부 렌더링

조건부 렌더링

- **조건부 렌더링**
  - 조건부 렌더링은 어떠한 조건에 따라서 렌더링이 달라지는 것

```
function UserGreeting(props) {
    return <h1> 다시 오셨군요 ! </h1>;
}

function GuestGreeting(props) {
    return <h1> 회원가입을 해주세요.</h1>;
}
```

```
function Greeting(props) {

    const isLoggedIn = props.isLoggedIn;

    if (isLoggedIn){
        return <UserGreeting />;
    }

    return <GuestGreeting />;
}
```

```
function App() {
  return (
    <Greeting isLoggedIn={false}/>
  );
}
```

조건부 렌더링

- 엘리먼트 변수
  - 렌더링해야 될 컴포넌트를 변수처럼 다루어야 할 때 사용할 수 있는 방법이 바로 엘리먼트 변수.

```
function LoginButton(props) {
    return (
        <button onClick={props.onClick}>
            로그인
        </button>
    );
}

function LogoutButton(props) {
    return (
        <button onClick={props.onClick}>
            로그아웃
        </button>
    );
}
```

```
function LoginControl(props) {

    const [isLoggedIn, setIsLoggedIn] = useState(false);

    const handleLoginClick = () => {
        setIsLoggedIn(true);
    }
    const handleLogoutClick = () => {
        setIsLoggedIn(false);
    }

    let button;

    if (isLoggedIn){
        button = <LogoutButton onClick={handleLogoutClick} />;
    } else {
        button = <LoginButton onClick={handleLoginClick} />;
    }

    return(
        <div>
        {button}
        </div>
    )
}
```

조건부 렌더링

- 인라인 조건
  - 코드를 별도로 분리된 곳에 작성하지 않고 해당 코드가 필요한 곳 안에 조건문을 코드 안에 넣어 처리
  - 인라인 If
    - 인라인 If if 문을 필요한 곳에 직접 집어 넣어서 사용하는 방법
    - 다만 실제로 if문을 넣는 것은 아니고, if문과 동일한 효과를 내기 위해 && 라는 논리 연산자를 사용
    - && 연산자는 흔히 AND 연산 이라고 부르는데 양쪽에 나오는 조건문이 모두 true인 경우에만 전체 결과가 true가 된다.

```
function Mailbox(props) {
    const unreadMessages = props.unreadMessages;

    return (
        <div>
            <h1> 안녕하세요!</h1>
            { unreadMessages.length> 0 &&
                <h2>
                 현재 {unreadMessages.length}개의 읽지 않은 메시지가 있습니다.
                </h2>
            }
        </div>
    );
}
```

```
const list = [{}, {}, {}, {}, {}]

function App() {
  return (

    <Mailbox unreadMessages={list}/>
  );
}
```

조건부 렌더링

- 인라인 조건
  - 인라인 If-else
    - 인라인 If-Else는 조건문의 값에 따라 다른 엘리먼트를 보여줄 때 사용
    - 삼항연산자를 사용해서 처리

```
import React from "react";

function UserStatus(props) {

    return (
        <div>
            이 사용자는 현재 <b>
        {props.isLoggedIn ? '로그인' : '로그인하지 않은 '}</b> 상태입니다.
        </div>
    )
}

export default UserStatus;
```

조건부 렌더링

- 인라인 조건
  - 인라인 If-else
    - 인라인 If-Else는 조건문의 값에 따라 다른 엘리먼트를 보여줄 때 사용
    - 삼항연산자를 사용해서 처리

```
/* if (isLoggedIn) {
    button = <LogoutButton onClick={handleLogoutClick} />;
} else {
    button = <LoginButton onClick={handleLoginClick} />;
} */

return (
    <div>
        <Greeting isLoggedIn={isLoggedIn} />
        {isLoggedIn ? <LogoutButton onClick={handleLogoutClick} />
                    : <LoginButton onClick={handleLoginClick} />
        }
    </div>
)
```

조건부 렌더링

- **컴포넌트 렌더링 막기**
  - 컴포넌트를 렌더링하고 싶지 않을 때에는 null을 리턴하면 됨.

```
import React from "react";

function WarningBanner(props) {

    if (!props.warning) {
        return null;
    }

    return (
        <div>경고!</div>
    );

}

export default WarningBanner;
```

```
function App() {
  return (
    <WarningBanner warning={true} />
  );
}
```

조건부 렌더링

- 컴포넌트 렌더링 막기
  - 컴포넌트를 렌더링하고 싶지 않을 때에는 null을 리턴하면 됨.

```
import React, { useState } from "react";
import WarningBanner from "./WarningBanner";

function MainPage(props) {

    const [showWarning, setShowWarning] = useState(false);

    const handleToggleClick = () => {
        setShowWarning(prevShowWarning => !prevShowWarning);
    }

    return (
        <div>
            <WarningBanner warning={showWarning} />
            <button onClick={handleToggleClick}>
                {showWarning ? '감추기' : '보이기'}
            </button>
        </div>
    )
}
export default MainPage;
```

```
function App() {
  return (
    <MainPage/>
  );
}
```

로그인 여부 처리하는 툴바

- Toolbar.jsx

```
const styles = {
    wrapper: {
        padding: 16,
        display: "flex",
        flexDirection: "row",
        borderBottom: "1px solid grey",
    },
    greeting: {
        marginRight: 8,
    },
};

function Toolbar(props) {
    const { isLoggedIn, onClickLogin, onClickLogout } = props;
    return (
        <div style={styles.wrapper}>
            {isLoggedIn && <span style={styles.greeting}>환영합니다!</span>}
            {isLoggedIn ? (
                <button onClick={onClickLogout}>로그아웃</button>
            ) : (
                <button onClick={onClickLogin}>로그인</button>
            )}
        </div>
    );
}

export default Toolbar;
```

로그인 여부 처리하는 툴바

- LandingPage.jsx

```jsx
import React, { useState } from "react";
import Toolbar from "./Toolbar";

function LandingPage(props) {
    const [isLoggedIn, setIsLoggedIn] = useState(false);

    const onClickLogin = () => {
        setIsLoggedIn(true);
    };

    const onClickLogout = () => {
        setIsLoggedIn(false);
    };

    return (
        <div>
            <Toolbar
                isLoggedIn={isLoggedIn}
                onClickLogin={onClickLogin}
                onClickLogout={onClickLogout}
            />
            <div style={{ padding: 16 }}>소플과 함께하는 리액트 공부!</div>
        </div>
    );
}

export default LandingPage;
```

```jsx
function App() {
  return (
    <LandingPage/>
  );
}
```

first
coding

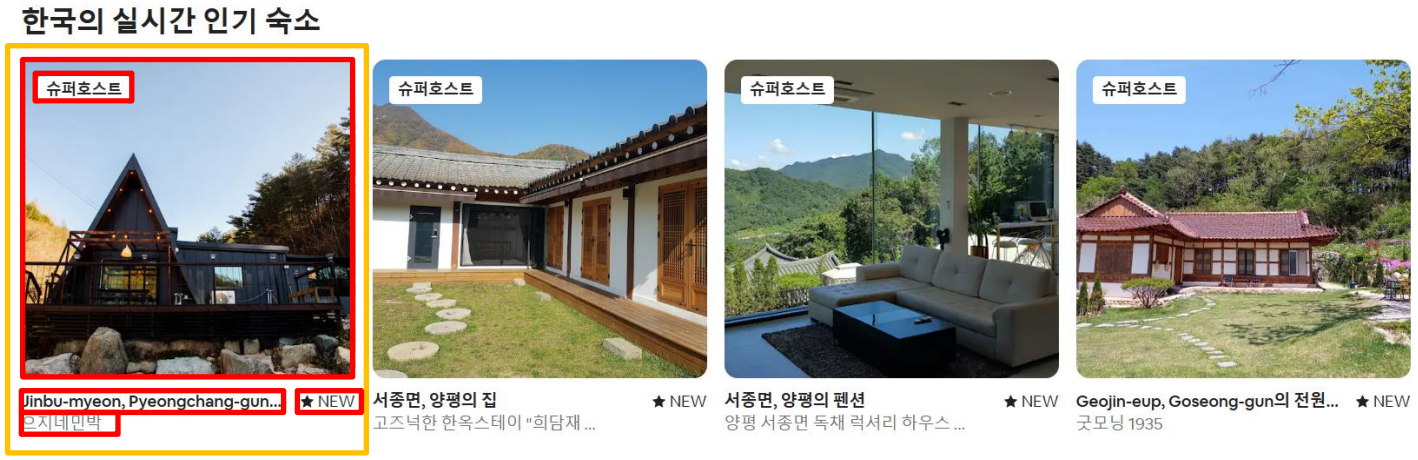
리스트와 키

리스트와 키

- 리스트와 키
  - 리스트를 위해 사용하는 자료구조가 배열
  - key
    - 컴퓨터 프로그래밍에서의 키는 각 객체나 아이템을 구분할 수 있는 고유한 값을 의미
  - 리액트에서는 배열과 키를 사용하여 반복되는 다수의 엘리먼트를 쉽게 렌더링할 수 있다.

리스트와 키

- 리스트와 키
  - 리스트를 위해 사용하는 자료구조가 배열
  - key
    - 컴퓨터 프로그래밍에서의 키는 각 객체나 아이템을 구분할 수 있는 고유한 값을 의미
  - 리액트에서는 배열과 키를 사용하여 반복되는 다수의 엘리먼트를 쉽게 렌더링할 수 있다.



리스트와 키

- **여러 개의 컴포넌트 렌더링**
  - 같은 컴포넌트를 화면에 반복적으로 나타내야 할 경우 자바스크립트 배열의 map() 함수를 이용하여 처리
    - 배열에 들어있는 각 변수에 필요한 데이터 처리 후 리턴 ➔ 엘리먼트를 반환

```jsx
import React from "react";

function NumberList(props) {

    const numbers = [1, 2, 3, 4, 5];
    const listItems = numbers.map((number) =>
        <li>{number}</li>
    )

    return (
        <ul>
            {listItems}
        </ul>
    )
}

export default NumberList
```

```jsx
function App() {
  return (
    <NumberList/>
  );
}
```

리스트와 키

- 기본적인 리스트 컴포넌트
  - NumberList 컴포넌트는 props로 숫자가 들어있는 배열인 numbers를 받아서 이를 목록으로 출력
  - 개발자 도구의 콘솔 탭에 리스트 아이템에는 무조건 키가 있어야 한다는 경고 문구가 나온다.

```jsx
import React from "react";

function NumberList(props) {

    //const numbers = [1, 2, 3, 4, 5];
    const { numbers } = props;


    const listItems = numbers.map((number) =>
        <li>{number}</li>
    )

    return (
        <ul>
            {listItems}
        </ul>
    )
}

export default NumberList
```

```jsx
const numbers = [1, 2, 3, 4, 5];

function App() {
  return (
    <NumberList numbers={numbers}/>
  );
}
```

```
▶Warning: Each child in a list should          react-jsx-dev-runtime.development.js:87
have a unique "key" prop.

Check the render method of `NumberList`.
See https://reactjs.org/link/warning-keys
for more information.
    at li
    at NumberList (http://localhost:3000/
static/js/bundle.js:904:5)
    at App
```

리스트와 키

- 리스트에서 사용하는 키
  - 리액트에서의 키는 리스트에서 아이템을 구분하기 위한 고유한 문자열
    - 키는 리스트에서 어떤 아이템이 변경, 추가 또는 제거되었는지 구분하기 위해 사용
    - 리액트에서의 키의 값은 같은 리스트에 있는 엘리먼트 사이에서만 고유한 값이면 된다
  - 키값의 생성 방법
    - 중복되지 않는 숫자
    - 값으로 인덱스index를 사용하는 방법 (아이템들의 고유한 id값이 없을 때 사용)
    - 아이템들의 고유한 id

```
const listItems = numbers.map((number, index) =>
    <li key={index}>{number}</li>
)
```

선수 리스트

- PlayerList.jsx

```jsx
import React from "react";

const players = [
    { id: 1, name: "손흥민", },
    { id: 2, name: "이강인", },
    { id: 3, name: "황희찬", },
    { id: 4, name: "김민재", },
];

function PlayerList(props) {
    return (
        <ul>
            {players.map((player, index) => {
                return <li key={player.id}>{player.name}</li>;
            })}
        </ul>
    );
}

export default PlayerList;
```

```jsx
function App() {
  return (
    <PlayerList/>
  );
}
```

first coding

Form

폼

- **제어 컴포넌트**
  - 제어 컴포넌트는 사용자가 입력한 값에 접근하고 제어할 수 있도록 해주는 컴포넌트
  - 리액트의 통제를 받는 컴포넌트
    - 제어 컴포넌트는 그 값이 리액트의 통제를 받는 입력 폼 엘리먼트
    - 모든 데이터를 state에서 관리
    - state의 값을 변경할 때에는 무조건 setState () 함수를 사용
    - 함수 컴포넌트에서는
    - useState( ) 훅을 사용하여 state를 관리

폼

- 제어 컴포넌트

```
<form>
    <label>
    이름:
    <input type="text" name="name" />
    </label>
    <button type="submit">제출</button>
</form>
```

```
function App() {
  return (
    <NameForm/>
  );
}
```

```
function NameForm(props) {

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

    const handleChange = (event) => {
        setValue(event.target.value);
    }
    const handleSubmit = (event) => {
        alert(' 입력한 이름: ' + value);
        event.preventDefault();
    }

    return (
        <form onSubmit={handleSubmit}>
            <label>
                이름:
                <input typea="text" value={value} onChange={handleChange} />
            </label>
            <button type="submit">제출</button>
        </form>
    )
}

export default NameForm;
```

폼

- textarea

```
<textarea>
안녕하세요, 여기에 이렇게 텍스트가 들어가게 됩니다.
</textarea>
```

```
function App() {
  return (
    <RequestForm/>
  );
}
```

```
function RequestForm(props) {

    const [value, setValue] = useState('요청사항을 입력하세요.');

    const handleChange = (event) => {
        setValue(event.target.value);
    }
    const handleSubmit = (event) => {
        alert(' 입력한 요청사항 : ' + value);
        event.preventDefault();
    }

    return (
        <form onSubmit={handleSubmit}>
            <label>요청사항:
                <textarea value={value} onChange={handleChange} />
            </label>
            <button type="submit">제출</button>
        </form>
    )
}

export default RequestForm;
```

폼

- select

```
<select>
<option value="apple">사과</option>
<option value="banana">바나나</option>
<option selected value="grape">포도
</option>
<option value="watermelon">수박
</option>
</select>
```

```
function App() {
  return (
    <Fruitselect/>
  );
}
```

```
function Fruitselect(props) {
    const [value, setValue] = useState('grape');
    const handleChange = (event) => {
        setValue(event.target.value);
    }
    const handleSubmit = (event) => {
        alert('선택한 과일:' + value);
        event.preventDefault();
    }
    return (
        <form onSubmit={handleSubmit}>
            <label>
                과일을 선택하세요 :
                <select value={value} onChange={handleChange}>
                    <option value="apple">사과</option>
                    <option value="banana">바나나 </option>
                    <option value="grape">포도</option>
                    <option value="watermelon">수박 </option>
                </select>
            </label>
            <button type="submit">제출</button>
        </form>
    )
}
export default Fruitselect;
```

폼

- 여러 개의 입력

```
function App() {
  return (
    <Reservation/>
  );
}
```

```
function Reservation(props) {
    const [haveBreakfast, setHaveBreakfast] = useState(true);
    const [numberOfGuest, setNumberOfGuest] = useState(2);
    const handleSubmit = (event) => {
        alert(`아침식사 여부: ${haveBreakfast}, 방문객 수: ${numberOfGuest}`);
        event.preventDefault();
    }
    return (
        <form onSubmit={handleSubmit}>
            <label>
                아침식사 여부:
                <input
                    type="checkbox"
                    checked={haveBreakfast}
                    onChange={(event) => {
                        setHaveBreakfast(event.target.checked);
                    }} />
            </label>
            <br />
            <label>
                방문객 수:
                <input
                    type="number"
                    value={numberOfGuest}
                    onChange={(event) => {
                        setNumberOfGuest(event.target.value);
                    }} />
            </label>
            <button type="submit">제출</button>
        </form>
    )
}
export default Reservation;
```

사용자 정보 입력

- SignUp.jsx

```jsx
function App() {
  return (
    <Fruitselect/>
  );
}
```

```jsx
function SignUp(props) {
    const [name, setName] = useState("");
    const [gender, setGender] = useState("남자");

    const handleChangeName = (event) => {
        setName(event.target.value);
    };
    const handleChangeGender = (event) => {
        setGender(event.target.value);
    };
    const handleSubmit = (event) => {
        alert(`이름: ${name}, 성별: ${gender}`);
        event.preventDefault();
    };

    return (
        <form onSubmit={handleSubmit}>
            <label>
                이름:
                <input type="text" value={name} onChange={handleChangeName} />
            </label>
            <br />
            <label>
                성별:
                <select value={gender} onChange={handleChangeGender}>
                    <option value="남자">남자</option>
                    <option value="여자">여자</option>
                </select>
            </label>
            <button type="submit">제출</button>
        </form>
    );
}

export default SignUp;
```