

React Rendering

조건부 렌더링

if문

null

삼항 연산자 (?:)

논리 연산자 (&&)

변수 선언

리스트 렌더링

배열을 데이터로 렌더링

배열 항목 필터링

key 속성을 사용하여 리스트 항목 순서 유지

key 규칙

key가 필요한 이유

조건부 렌더링

Component는 조건에 따라 다른 항목을 표시해야 하는 경우가 많다. React는 `if` 문, `&&`, `?:` 연산자와 같은 JavaScript 문법을 사용하여 조건부로 JSX를 렌더링 할 수 있다.

if문

```
1 function Item({ name, isPacked }) {  
2   if (isPacked) {  
3     return <li className="item">{name} ✓</li>;  
4   }  
5   return <li className="item">{name}</li>;  
6 }
```

null

조건부로 null을 반환하면 아무것도 렌더링하지 않을 수 있다.

```
1 function Item({ name, isPacked }) {  
2   if (isPacked) {  
3     return null;  
4   }  
5   return <li className="item">{name}</li>;  
6 }
```


삼항 연산자 (?:)

```
1 if (isPacked) {  
2   return <li className="item">{name} ✓</li>;  
3 }  
4 return <li className="item">{name}</li>;
```

위 코드는 정상적으로 동작하지만 반복되는 코드가 존재한다. 이런 코드는 삼항 연산자로 조금 더 드라이하게 만들 수 있다.

```
1 return (
```

```

2 <li className="item">
3   {isPacked ? name + '  ': name}
4 </li>
5 );


```

다만, 구문이 간단할 때 사용하는 것이 좋다.

논리 연산자 (&&) [↗](#)

React Components에서 조건이 참일 때 일부 JSX를 렌더링하고 그렇지 않으면 아무것도 렌더링하지 않는 경우가 많다. 이런 경우 논리 연산자를 활용할 수 있다. 아래 코드는 `isPacked` 가 `true` 면 체크 표시를 렌더링하고 그렇지 않으면 아무것도 렌더링하지 않겠다는 의미를 전달한다.

```


1 return (
2   <li className="item">
3     {name} {isPacked && '  '}
4   </li>
5 );

```

변수 선언 [↗](#)

단순하게 텍스트를 바꾸는 일이라면 변수를 조건에 따라 다르게 선언하는 방법도 있다. 이 방법은 장황하지만 가장 유연한 코드를 제공한다.

```

1 function Item({ name, isPacked }) {
2   let itemContent = name;
3   if (isPacked) {
4     itemContent = name + "  ";
5   }
6   return (
7     <li className="item">
8       {itemContent}
9     </li>
10  );
11 }

```

리스트 렌더링 [↗](#)

JavaScript 배열 메서드를 사용하여 데이터 배열을 조작할 수 있다. React에서 `filter()` 와 `map()` 을 사용해 데이터를 필터링하고 컴포넌트 배열로 변환할 수 있다. 먼저 아래와 같은 리스트가 있다고 가정하자

```

1 <ul>
2   <li>Creola Katherine Johnson: mathematician</li>
3   <li>Mario José Molina-Pasquel Henríquez: chemist</li>
4   <li>Mohammad Abdus Salam: physicist</li>
5   <li>Percy Lavon Julian: chemist</li>
6   <li>Subrahmanyan Chandrasekhar: astrophysicist</li>
7 </ul>

```

배열을 데이터로 렌더링 [↗](#)

1. 먼저 데이터를 배열로 만든다.

```

1 const people = [
2   'Creola Katherine Johnson: mathematician',
3   'Mario José Molina-Pasquel Henríquez: chemist',

```

```

4 'Mohammad Abdus Salam: physicist',
5 'Percy Lavon Julian: chemist',
6 'Subrahmanyam Chandrasekhar: astrophysicist'
7 ];

```

2. 배열의 요소를 새로운 JSX 노드 배열에 매핑한다.

```

1 const listItems = people.map(person => <li>{person}</li>);

```

3. `` 로 래핑된 컴포넌트에 `listItems` 를 반환한다.

```

1 return <ul>{listItems}</ul>;

```

4. 결과 이미지



배열 항목 필터링 ↻

예제에 있는 사람들 중 특정 직업군만 표시하고 싶다고 가정하자. JavaScript의 `filter()` 메서드를 사용하여 해당하는 사람만 반환할 수 있다.

```

1 const people = [{
2   id: 0,
3   name: 'Creola Katherine Johnson',
4   profession: 'mathematician',
5 }, {
6   id: 1,
7   name: 'Mario José Molina-Pasquel Henríquez',
8   profession: 'chemist',
9 }, {
10  id: 2,
11  name: 'Mohammad Abdus Salam',
12  profession: 'physicist',
13 }, {
14  id: 3,
15  name: 'Percy Lavon Julian',
16  profession: 'chemist',
17 }, {
18  id: 4,
19  name: 'Subrahmanyam Chandrasekhar',
20  profession: 'astrophysicist',
21 }];

```

1. `people` 에 `filter()` 를 적용하여 반환된 배열을 새로운 변수에 할당한다.

```

1 const chemists = people.filter(person =>
2   person.profession === 'chemist'
3 );

```

2. `chemists` 를 매핑한다.

```

1  const listItems = chemists.map(person =>
2    <li>
3      <img
4        src={getImageUrl(person)}
5        alt={person.name}
6      />
7      <p>
8        <b>{person.name}</b>
9        {' ' + person.profession + ' '}
10       known for {person.accomplishment}
11     </p>
12   </li>
13 );

```

3. `listItems` 를 반환한다.

```

1  return <ul>{listItems}</ul>;

```

4. 결과 이미지



❗ 람다식(`=>`)은 바로 뒤에 식을 반환하기 때문에 `return` 문이 필요하지 않다.

```

1  const listItems = chemists.map(person =>
2    <li>...</li> // 암시적 반환!
3  );

```

그러나, `=>` 뒤에 중괄호가 오는 경우 명시적으로 `return` 을 작성해야 한다.

```

1  const listItems = chemists.map(person => { // 중괄호
2    return <li>...</li>;
3  });

```

key 속성을 사용하여 리스트 항목 순서 유지 ↻

위 예제를 IDE에서 진행하면 `key prop` 을 할당하라는 경고 표시가 발생한다. `map()` 호출 내부에 JSX Component에는 항상 `key` 가 필요하다. `key` 는 각 컴포넌트가 어떤 배열 항목에 해당하는지 React에 알려줌으로써 추후 DOM 트리에서 업데이트가 발생했을 때 올바르게 빠른 업데이트가 가능하도록 한다. (즉석에서 `key` 를 생성하는 것 보다 데이터에 `key` 속성을 추가하는 것이 일반적이다.)

```

1  <li key={person.id}>...</li>

```

만약, 여러 컴포넌트를 하나의 부모 컴포넌트로 묶어야 하는 경우라면 `Fragment` 를 사용할 수 있는데 단순 `<>` 태그는 `id` 속성을 부여할 수 없기 때문에 `<Fragment>` 로 선언해야 한다. (마찬가지로 `<Fragment>` 태그는 브라우저에서 사라진다.)

```

1  import { Fragment } from 'react';
2
3  // ...

```

```
4
5  const listItems = people.map(person =>
6    <Fragment key={person.id}>
7      <h1>{person.name}</h1>
8      <p>{person.bio}</p>
9    </Fragment>
10  );
```

key 규칙 [↗](#)

- 같은 배열(같은 데이터셋) 안에서 key는 고유한 값이어야 한다.
- key는 변경되어서는 안된다.

key가 필요한 이유 [↗](#)

데이터 셋 안에서 해당 값을 식별하기 위해서 사용한다. key를 사용해서 특정 데이터에 빠르게 접근할 수 있고 DOM 트리 안에서 변경된 사항을 쉽게 업데이트할 수 있다. React는 해당 컴포넌트 생명주기 내내 key 값으로 컴포넌트를 식별할 수 있다.

참고 자료 :

[⚙️ 조건부 렌더링 - React](#)

[⚙️ 리스트 렌더링 - React](#)