

# React

map, filter, 단축평가

# map()

# 일반 for 반복문

```
let list = ['a', 'b', 'c', 'd', 'e'];  
for ( let i = 0; i < list.length; i++ ) {  
  |   console.log( list[i] );  
}  
}
```

# map() 함수

- map()의 인자로 넘겨지는 callback 함수를 실행한 결과를 가지고 새로운 배열을 만들 때 사용.
- map() 함수를 필요에 따라 반복문처럼 사용할 수도 있음.

코드와 함께 알아보자!

# map() 함수 문법

```
arr.map( callbackFunction, [thisArg] )
```

- callbackFunction
  - 새로운 배열의 요소를 생성하는 함수로, currentValue, index, array 3개의 인수를 가질 수 있다.
- [this.Arg] 는 생략 가능한 것으로 callbackFunction 에서 사용할 this 객체

# map() 함수

```
let list = ['a', 'b', 'c', 'd', 'e'];  
let items = list.map((txt, id, arr) => {  
  console.log("txt: ", txt);  
  console.log("id: ", id);  
  console.log("arr: ", arr);  
  return txt + id;  
})
```

```
items ▼ (5) ['a0', 'b1', 'c2', 'd3', 'e4'] ⓘ  
  0: "a0"  
  1: "b1"  
  2: "c2"  
  3: "d3"  
  4: "e4"  
  length: 5  
  ▶ [[Prototype]]: Array(0)
```

# map() 함수

```
let list = ['a', 'b', 'c', 'd', 'e'];  
let items = list.map((txt, id, arr)=>{  
  console.log("txt: ", txt);  
  console.log("id: ", id);  
  console.log("arr: ", arr);  
  return txt + id;  
})
```

- txt : list 를 순서대로 돌면서 나오는 값
- id : 방금 나온 값(txt)의 인덱스
- arr : 현재 반복을 돌고 있는 배열
- items : “return txt + id;” 로 만들어진 배열

# 배열 데이터, map()과 함께 사용

- 배열 데이터를 좀 더 효율적으로 그리기 위해서 map 사용!

```
{배열}.map((요소, 인덱스) => {  
    return <div key={인덱스}>{요소}</div>;  
})}
```

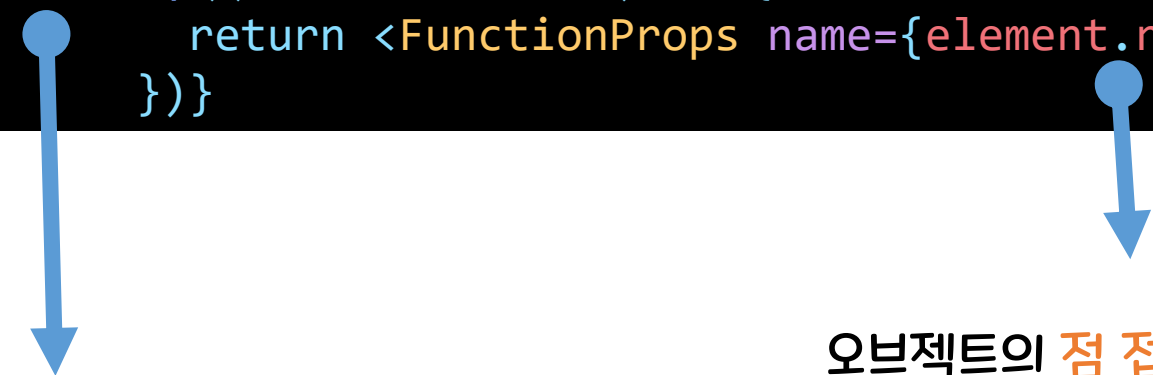
- key
  - 기존 요소와 업데이트 요소를 비교하는데 사용되는 속성,
  - 다른 요소와 겹치지 않는 **고유한 값**이어야 합니다.
  - key는 고유한 값을 가져야 하기 때문에 배열의 요소 중 고유한 값(id 등)이 존재하지 않는다면 index로 사용해도 됩니다! (단, index 를 사용하는 것은 최후의 수단!)



# 배열 데이터, map()과 함께 사용

- div 같은 태그가 아닌 만들어진 **컴포넌트**와 함께 사용할 수도 있겠죠!

```
{배열.map((element, index) => {  
    return <FunctionProps name={element.name} key={index} />;  
})}
```



```
let 배열=[{name:"allie"},  
          {name:"Lucy"},  
          {name:"Linda"},  
          ];
```

와 같은 **오브젝트로 이루어진 배열**이라면

오브젝트의 **점 접근법**으로 props 전달 &  
배열의 개수만큼 컴포넌트 생성,

# Component에 map() 적용

```
function App() {  
  const list = ["k", "d", "t", "w", "e", "b"];  
  
  return (  
    <>  
      <ol>  
        {list.map((value, idx) => {  
          return <li key={idx}>{value}</li>;  
        })}  
      </ol>  
    </>  
  );  
}  
  
export default App;
```

1. k
2. d
3. t
4. w
5. e
6. b

# Component에 map() 적용

```
⊗ ▶Warning: Each child in a list      react-refresh-runtime.development.js:688  
   should have a unique "key" prop.  
  
Check the render method of `MapTest`. See https://reactjs.org/link/warning-  
  keys for more information.  
    at li  
    at MapTest (http://localhost:3000/main.f503859...hot-update.js:30:74)  
    at App
```

- map() 함수를 이용해 컴포넌트를 생성할 때 “key” 사용을 권장한다.
- Why? React는 자율적으로 업데이트 전 기존 요소와 업데이트 요소를 비교하는데 key를 사용한다.

# Component에 map() 적용

```
<li key={id}>{value}</li>
```

- Key를 index 값으로 설정할 시, 리스트의 순서가 변경되면 모든 key가 변경되므로 key는 index 가 아닌 고유한 값으로 설정해야 한다
  - But, 현재는 **고유 값으로 설정할 만한 게 없으니, index로 테스트**

# Component에 map() 적용

- 각 원소마다 고유 id 값을 갖고 있다면? 다음과 같이 설정할 수 있다!

```
const list = [  
  {  
    id: 1,  
    alpha: "a",  
  },  
  {  
    id: 2,  
    alpha: "b",  
  },  
  {  
    id: 3,  
    alpha: "c",  
  },  
  {  
    id: 4,  
    alpha: "d",  
  },  
  {  
    id: 5,  
    alpha: "e",  
  },  
];
```

```
return (  
  <>  
    <ol>  
      {list.map((value) => {  
        return <li key={value.id}>{value.alpha}</li>;  
      })}  
    </ol>  
  </>  
)
```

# filter()

# filter() 함수

- filter()의 인자로 넘겨지는 callback 함수의 **테스트(조건)를 통과하는 요소**를 모아 새로운 배열을 생성.
- filter() 함수를 사용하여 배열에서 원하는 값을 삭제하는 코드 구현 가능.

코드와 함께 알아보자!

# filter() 함수

```
let animals = ['dog', 'cat', 'rabbit'];  
  
let newAnimals = animals.filter((animal)=>{ return animal.length > 3});  
console.log(newAnimals);
```

▶ `['rabbit']`

```
let newAnimals = animals.filter((animal)=> animal.length > 3);
```



# filter() 함수

```
let words = ['dog', 'cat', 'rabbit'];  
  
let result2 = words.filter((word) => {  
  return word.includes('a');  
});  
console.log( result2 );
```

# 단축평가

# 단축평가란?

- 논리 연산자를 사용하여 특정 조건에 따라 값을 결정하거나, 조건에 따라 특정 코드를 실행하는 방법

## 1) && 연산자를 사용한 단축 평가

- 둘 다 참

## 2) || 연산자를 사용한 단축 평가

- 둘 중 하나 참

# && 연산자

## A && B

- A가 false : B는 아예 확인하지 않고 바로 A의 값을 반환합니다. A가 이미 false이므로 A와 B 모두 참일 수 없기 때문

```
const result = false && "Hello";  
console.log(result); // 출력: false
```

- A가 true: 만약 A가 true 라면, B의 값을 확인. 이 경우, B의 값이 반환됩니다.

```
const name = "Martin";  
const greeting = name && `Hello, ${name}!`;   
console.log(greeting); // 출력: "Hello, Martin!"
```

# || 연산자

## A || B

- A가 false: B의 값을 확인해야 합니다. 이 경우, B의 값이 반환됩니다.

```
const defaultName = "Martin";  
const userName = null;  
const displayName = userName || defaultName;  
console.log(displayName); // 출력: "Martin"
```

- A가 true: B는 아예 확인하지 않고 바로 A의 값을 반환합니다. 왜냐하면 A가 이미 true이므로 A나 B 중 하나만 참이면 되기 때문입니다.

```
const result = true || "Hello";  
console.log(result); // 출력: true
```