map, filter, reduce메서드

iterable.map

자신을 호출한 배열의 모든 요소를 순회하면서 인수로 전달받은 콜백 함수를 반복 호출한다.

그리고 **콜백함수의 반환값들로 구성된 새로운 배열을 반환**한다.

> 💡 **※일반 적인 연산 처리를 할 때에는 forEach메서드를 사용한다.**

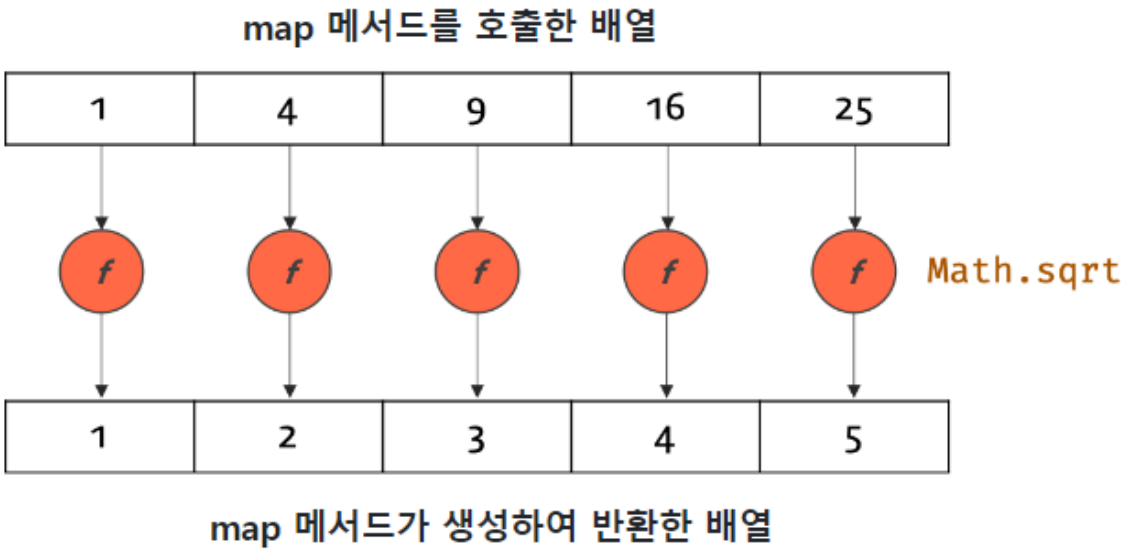
```
const numbers = [1, 4, 9];

// map 메서드는 numbers 배열의 모든 요소를 순회하면서 콜백 함수를 반복 호출한다.
// 그리고 콜백 함수의 반환값들로 구성된 새로운 배열을 반환한다.
const roots = numbers.map(item => Math.sqrt(item));

// 위 코드는 다음과 같다.
// const roots = numbers.map(Math.sqrt);

// map 메서드는 새로운 배열을 반환한다
console.log(roots);   // [ 1, 2, 3 ]
// map 메서드는 원본 배열을 변경하지 않는다
console.log(numbers); // [ 1, 4, 9 ]

//map 메서드가 생성하여 반환하는 새로운 배열의 length 프로퍼티 값은 map 메서드를 호출한 배열의 length 프로퍼티 값과 반드시 일치한다.
//즉, map 메서드를 호출한 배열과 map 메서드가 생성하여 반환한 배열은 1:1 매핑한다.
```



forEach 메서드와 마찬가지로 map 메서드도 map 메서드를 호출한 배열의 요소값과 인덱스 그리고 map메서드를 호출한 배열(this)을 순차적으로 인수로 전달한다.

```
class Prefixer {
  constructor(prefix) {
    this.prefix = prefix;
  }

  add(arr) {
    // 화살표 함수 내부에서 this를 참조하면 상위 스코프의 this를 그대로 참조한다.
    return arr.map(item => this.prefix + item);
  }
}

const prefixer = new Prefixer('-webkit-');
console.log(prefixer.add(['transition', 'user-select']));
// ['-webkit-transition', '-webkit-user-select']
```

연습 문제

문제 1: 점수 변환

> `scores` 배열의 각 점수를 `0`과 `1` 사이의 비율로 변환하여 새로운 배열을 만드세요

```
const scores = [50, 75, 100, 25, 0];

// 출력
// [0.5, 0.75, 1, 0.25, 0]
```

💡 **힌트:** 점수 비율은 `score / 100`으로 계산할 수 있습니다.

답

문제 2: 문자열 길이 배열 만들기

> 주어진 문자열 배열의 각 문자열에 대해 그 길이를 계산하여 새로운 배열을 생성하세요
>
> `strings` 배열이 각 문자열의 길이를 계산하여 새로운 배열을 만드세요

```
const strings = ["apple", "banana", "cherry"];

// 출력
// [5, 6, 6]
```

💡 **힌트:** 문자열 길이는 `string.length`로 계산할 수 있습니다

답

문제 3: 객체 배열에서 특정 속성 추출하기

> 각 객체의 `name` 속성만을 추출하여 새로운 배열을 생성하세요

```
const users = [
  { name: "Alice", age: 25 },
  { name: "Bob", age: 30 },
  { name: "Charlie", age: 35 }
];

// 출력
// ["Alice", "Bob", "Charlie"]
```

답

---

iterable.filter

filter 메서드는 자신을 호출한 배열의 모든 요소를 순회하면서 인수로 전달받은 콜백함수를 반복 호출한다.

콜백 함수의 반환값이 true인 요소로만 구성된 새로운 배열을 반환한다.

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

// filter 메서드는 numbers 배열의 모든 요소를 순회하면서 콜백 함수를 반복 호출한다.
// 그리고 콜백 함수의 반환값이 true인 요소로만 구성된 새로운 배열을 반환한다.
// 다음의 경우 numbers 배열에서 홀수인 요소만을 필터링한다(1은 true로 평가된다).
const odds = numbers.filter(item => item % 2);
console.log(odds); // [1, 3, 5]

//filter 메서드는 자신을 호출한 배열에서 필터링 조건을 만족하는 특정 요소만 추출하여 새로운 배열을 만들고 싶을 때 사용한다.
```

filter 메서드는 자신을 호출한 배열에서 특정 요소를 제거하기 위해 사용할 수도 있다.

```
class Users {
  constructor() {
    this.users = [
      { id: 1, name: 'Lee' },
      { id: 2, name: 'Kim' }
    ];
  }
    // 요소 추출
  findById(id) {
    // id가 일치하는 사용자만 반환한다.
    return this.users.filter(user => user.id === id);
  }
  // 요소 제거
  remove(id) {
    // id가 일치하지 않는 사용자를 제거한다.
    this.users = this.users.filter(user => user.id !== id);
  }
}

const users = new Users();
let user = users.findById(1);
console.log(user); // [{ id: 1, name: 'Lee' }]

// id가 1인 사용자를 제거한다.
users.remove(1);

user = users.findById(1);
console.log(user); // []

//filter 메서드를 사용해 특정 요소를 제거할 경우 특정 요소가 중복되어 있다면 중복된 요소가 모두 제거된다.
//특정 요소를 하나만 제거하려면 indexOf 메서드를 통해 특정 요소의 인덱스를 취득한 다음 splice 메서드를 사용한다.

//중복 요소를 제거
```

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

// 현재 순회 중인 요소의 인덱스 i가 val의 인덱스와 같다면 val은 처음 순회하는 요소다. 이 요소만 필터링한다.
const result = values.filter((val, i, _values) => _values.indexOf(val) === i);
console.log(result); // [1, 2, 3, 5, 4]

//중복되지 않는 유일한 값들의 집합인 Set을 사용할 수도 있다.(이 방법을 추천)
const values = [1, 2, 1, 3, 5, 4, 5, 3, 4, 4];

// 중복을 허용하지 않는 Set 객체의 특성을 활용하여 배열에서 중복된 요소를 제거할 수 있다.
const result = [...new Set(values)];
console.log(result); // [1, 2, 3, 5, 4]
```

iterable.reduce

자신을 호출한 배열을 모든 요소를 순회하며 인수로 전달받은 콜백 함수를 반복 호출한다.

콜백 함수의 반환값을 다음 순회 시에 콜백 함수의 첫 번째 인수로 전달하면서 콜백 함수를 호출하여 **하나의 결과값을 만들어 반환**한다.

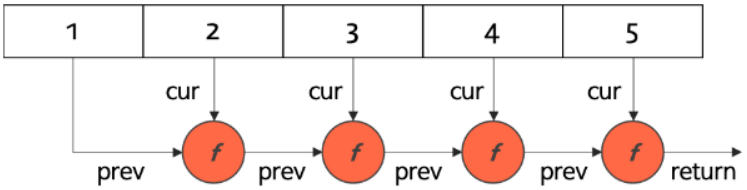
예제는 2개의 인수, 콜백함수와 초기값 0을 전달받아 자신을 호출한 배열의 모든 요소의 누적한 결과를 반환한다.

```
// [1, 2, 3, 4]의 모든 요소의 누적을 구한다.
const sum = [1, 2, 3, 4].reduce((accumulator, currentValue, index, array) => accumulator + currentValue, 0);

console.log(sum); // 10
```

reduce 메서드의 콜백 함수는 4개의 인수를 전달받아 배열의 length만큼 총 4회 호출된다. 이때 콜백 함수로 전달되는 인수와 콜백 함수의 반환값은 다음과 같다.

구분	콜백 함수에 전달되는 인수				콜백 함수의 반환값
	accumulator	currentValue	index	array	
첫 번째 순회	0 (초기값)	1	0	[1, 2, 3, 4]	1 (accumulator + currentValue)
두 번째 순회	1	2	1	[1, 2, 3, 4]	3 (accumulator + currentValue)
세 번째 순회	3	3	2	[1, 2, 3, 4]	6 (accumulator + currentValue)
네 번째 순회	6	4	3	[1, 2, 3, 4]	10 (accumulator + currentValue)



reduce 메서드는 자신을 호출한 배열의 모든 요소를 순회하며 하나의 결과값을 구해야 하는 경우에 사용한다.

```
//평균 구하기
const values = [1, 2, 3, 4, 5, 6];

const average = values.reduce((acc, cur, i, { length }) =>
{
  // 마지막 순회가 아니면 누적값을 반환하고 마지막 순회면 누적값으로 평
균을 구해 반환한다.
  return i === length - 1 ? (acc + cur) / length : acc + cu
r;
}, 0);

console.log(average); // 3.5

//최대값 구하기
const values = [1, 2, 3, 4, 5, 6];

const max = values.reduce((acc, cur) => (acc > cur ? acc :
cur), 0);
console.log(max); // 6

//Math.max 메서드를 사용하는 방법이 더 직관적이다.
```

```
const values = [1, 2, 3, 4, 5, 6];

const max = Math.max(...values);
// var max = Math.max.apply(null, values);
console.log(max); // 6

//요소의 중복 횟수 구하기
const fruits = ['banana', 'apple', 'orange', 'orange', 'app
le'];
const count = fruits.reduce((acc, cur) => {
  // 첫 번째 순회 시 acc는 초기값인 {}이고 cur은 첫 번째 요소인 'ban
ana'다.
  // 초기값으로 전달받은 빈 객체에 요소값인 cur을 프로퍼티 키로, 요소의
개수를 프로퍼티 값으로
  // 할당한다. 만약 프로퍼티 값이 undefined(처음 등장하는 요소)이면
프로퍼티 값을 1로 초기화한다.
  acc[cur] = (acc[cur] || 0) + 1;
  return acc;
}, {});
// 콜백 함수는 총 5번 호출되고 다음과 같이 결과값을 반환한다.
//{banana: 1} => {banana: 1, apple: 1} => {banana: 1, appl
e: 1, orange: 1}
//=> {banana: 1, apple: 1, orange: 2} => {banana: 1, apple:
2, orange: 2}
console.log(count); // { banana: 1, apple: 2, orange: 2 }
```