JavaScript

Collections(Array 확장) – 김근형 강사

Collection

- Collection
  - 값을 담을 수 있는 컨테이너
  - 프로그래밍 언어 대부분은 콜렉션을 제공하고 있다.
  - 자바 스크립트에서도 콜렉션을 제공하도록 ECMAScript6에서 패치가 되었으며 내용은 다음과 같다.
    - Indexed Collection - Arrays, Typed Array
    - Keyed Collection - Objects, Map, Set, Weak Map, Weak Set
  - ES5 때 까지는 Object와 Array만이 존재 하였으며 ES6 부터 Map, Set, WeakMap, WeakSet, Typed Array 가 추가되었다.

Array 확장

- Array 확장
  - 기존 Array 객체에서 ES6 버전에서 함수가 추가됨

함수	설 명
from()	유사 배열 또는 반복 가능한 객체로부터 새로운 Array 인스턴스를 생성한다.
of()	전달인자의 개수나 데이터 타입에 관계없이 새 Array 인스턴스를 생성한다
copyWithin()	배열 내의 지정된 요소들을 동일한 배열 내에서 복사한다
fill()	배열 안의 시작 인덱스부터 끝 인덱스까지의 요소 값을 지정된 정적 값으로 채운다
unshift()	배열의 앞에 하나 이상의 요소를 추가하고 새로운 길이를 반환한다.
includes()	배열에 특정 요소가 포함돼있는지 알아내어 true 또는 false를 적절히 반환한다.
indexOf()	배열에서 지정한 값과 같은 요소의 첫 인덱스를 반환합니다. 없으면 -1을 반환한다.
lastIndexOf()	배열에서 지정한 값과 같은 요소의 마지막 인덱스를 반환합니다. 없으면 -1을 반환한다.
toString()	배열과 요소를 반환하는 문자열을 반환한다. Object.prototype.toString() 메서드를 재정의
entries()	배열의 각 인덱스에 대한 키/값 쌍을 포함하는 새로운 배열 반복자 객체를 반환한다.
every()	만약 배열의 모든 요소가 제공된 검사 함수를 만족하면 true를 반환한다.

Array 확장

- Array 확장
  - 기존 Array 객체에서 ES6 버전에서 함수가 추가됨

함수	설 명
filter()	주어진 필터링 함수의 값의 결과가 참인 경우의 배열 요소들만으로 새로운 배열을 생성하여 반환한다
find()	주어진 테스팅 함수의 요구조건을 만족하는 배열 요소를 반환한다. 그러한 배열 요소가 없으면 undefined를 반환한다.
findIndex()	주어진 테스트 함수를 만족하는 배열의 첫 번째 요소에 대한 인덱스를 반환한다. 그렇지 않으면 -1이 리턴된다.
forEach()	배열의 각각의 요소에 함수를 호출한다.
keys()	배열의 각 인덱스에 대한 key들을 가지는 새로운 Array Iterator 객체를 반환한다.
map()	배열 내의 모든 요소 각각에 대하여 제공된 함수(callback)를 호출하고, 그 결과를 모아서 만든 새로운 배열을 반환한다.
reduce()	배열의 각 값에 대해 왼쪽에서 오른쪽으로 함수를 적용하여 단일 값으로 줄인다.
reduceRight()	배열의 각 값에 대해 오른쪽에서 왼쪽으로 함수를 적용하여 단일 값으로 줄인다.
some()	R배열중의 적어도 한 요소가 테스팅 함수를 만족시킨 다면 true를 반환합니다.
values()	배열의 요소값들에 대한 Array Iterator 객체를 반환합니다.

Array 확장

- from()
  - 다음과 같은 경우에 Array.from()으로 새Array를 만들 수 있다.
    - 유사 배열 객체 (length 속성과 인덱싱된 요소를 가진 객체)
    - 순회 가능한 객체 (Map, Set 등객체의 요소를 얻을 수 있는 객체)

```
// String에서 배열 만들기
let arr1 = Array.from('foo'); // ["f", "o", "o"]
console.log(arr1);
// Set에서 배열 만들기
const s = new Set(['foo', window]);
let arr2 = Array.from(s);
console.log(arr2);
// Map에서 배열 만들기
const m = new Map([[1, 2], [2, 4], [4, 8]]);
let arr3 = Array.from(m);
console.log(arr3); // [[1, 2], [2, 4], [4, 8]]
```

```
▶ (3) ["f", "o", "o"]
▶ (2) ["foo", Window]
▶ (3) [Array(2), Array(2), Array(2)]
```

Array 확장

- from()

```
// 배열 형태를 가진 객체(arguments)에서 배열 만들기
function f() {
  return Array.from(arguments);
}
console.log(f(1, 2, 3)); // [1, 2, 3]

// Array.from과 화살표 함수 사용하기
console.log(Array.from([1, 2, 3], x => x + x)); // [2, 4, 6]
console.log(Array.from({length: 5}, (v, i) => i)); // [0, 1, 2, 3, 4]

// 시퀀스 생성기(range)
const range = (start, stop, step) =>
Array.from({ length: (stop - start) / step + 1}, (_, i) => start + (i * step));
range(0, 4, 1); // [0, 1, 2, 3, 4]
range(1, 10, 2); // [1, 3, 5, 7, 9]
range('A'.charCodeAt(0), 'Z'.charCodeAt(0), 1).map(x => String.fromCharCode(x));
```

```
▶ (3) [1, 2, 3]                                                    VM285:5
▶ (3) [2, 4, 6]                                                    VM285:8
▶ (5) [0, 1, 2, 3, 4]                                              VM285:9
▶ (26) ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U",
  "V", "W", "X", "Y", "Z"]
```

Array 확장

- of()
  - Array.of() 메서드는 인자의 수나 유형에 관계없이 가변 인자를 갖는 새 Array 인스턴스를 만든다.
  - Array.of()와 Array 생성자의 차이는 정수형 인자의 처리 방법에 있다. Array.of(7)은 하나의 요소 7을 가진 배열을 생성하지만 Array(7)은 length 속성이 7인 빈 배열을 생성한다.

```
console.log(Array.of(7));       // [7]
console.log(Array.of(1, 2, 3)); // [1, 2, 3]

console.log(Array(7));          // [ , , , , , , ]
console.log(Array(1, 2, 3));    // [1, 2, 3]
```

```
▶ [7]
▶ (3) [1, 2, 3]
▶ (7) [empty × 7]
▶ (3) [1, 2, 3]
```

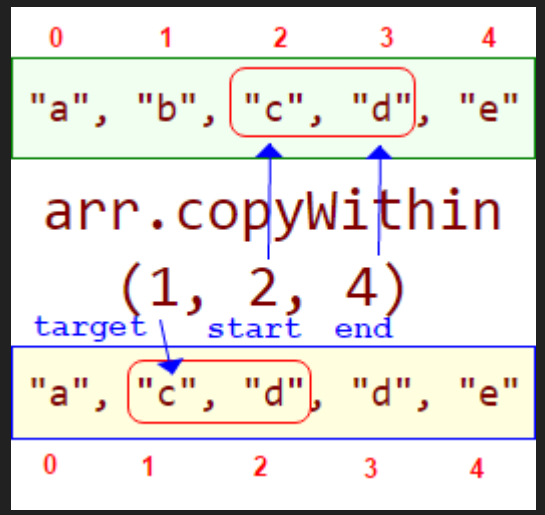
Array 확장

- copyWithin()
  - copyWithin() 메서드는 배열의 일부를 얕게 복사한 뒤, 동일한 배열의 다른 위치에 덮어쓰고 그 배열을 반환한다. 이 때, 크기(배열의 길이)를 수정하지 않고 반환한다.
  - 형식 : arr.copyWithin(target[, start[, end]])

매개변수명	설명
target	복사한 시퀀스(값)를 넣을 위치를 가리키는 0 기반 인덱스. 음수를 지정하면 인덱스를 배열의 끝에서부터 계산한다. target이 arr.length보다 크거나 같으면 아무것도 복사하지 않는다. target이 start 이후라면 복사한 시퀀스를 arr.length에 맞춰 자른다.
start	복사를 시작할 위치를 가리키는 0 기반 인덱스. 음수를 지정하면 인덱스를 배열의 끝에서부터 계산한다. 기본값은 0으로, start를 지정하지 않으면 배열의 처음부터 복사한다.
end	복사를 끝낼 위치를 가리키는 0 기반 인덱스. copyWithin은 end 인덱스 이전까지 복사하므로 end 인덱스가 가리키는 요소는 제외한다. 음수를 지정하면 인덱스를 배열의 끝에서부터 계산한다. 기본값은 arr.length로, end를 지정하지 않으면 배열의 끝까지 복사한다.

Array 확장

- copyWithin()




```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.copyWithin(2,0);
console.log(fruits);
var fruits = ["Banana", "Orange", "Apple", "Mango", "Kiwi", "Papaya"];
fruits.copyWithin(2,0,2);
console.log(fruits);
console.log([1, 2, 3, 4, 5].copyWithin(-2)); // [1, 2, 3, 1, 2]
console.log([1, 2, 3, 4, 5].copyWithin(0, 3)); // [4, 5, 3, 4, 5]
console.log([1, 2, 3, 4, 5].copyWithin(0, 3, 4)); // [4, 2, 3, 4, 5]
console.log([1, 2, 3, 4, 5].copyWithin(-2, -3, -1)); // [1, 2, 3, 3, 4]
console.log([].copyWithin.call({length: 5, 3: 1}, 0, 3)); // {0: 1, 3: 1, length: 5}
```

```
▶ (4) ["Banana", "Orange", "Banana", "Orange"]
▶ (6) ["Banana", "Orange", "Banana", "Orange", "Kiwi", "Papaya"]
▶ (5) [1, 2, 3, 1, 2]
▶ (5) [4, 5, 3, 4, 5]
▶ (5) [4, 2, 3, 4, 5]
▶ (5) [1, 2, 3, 3, 4]
▶ {0: 1, 3: 1, length: 5}
```

Array 확장

- fill()
  - 배열의 시작 인덱스부터 끝 인덱스의 이전까지 정적인 값 하나로 채웁니다.
  - 형식 : arr.fill(value[, start[, end]])

매개변수명	설명
value	배열을 채울 값.
start	시작 인덱스, 기본 값은 0.
end	끝 인덱스, 기본 값은 this.length.

Array 확장

- fill()

```
console.log([1, 2, 3].fill(4));               // [4, 4, 4]
console.log([1, 2, 3].fill(4, 1));            // [1, 4, 4]
console.log([1, 2, 3].fill(4, 1, 2));         // [1, 4, 3]
console.log([1, 2, 3].fill(4, 1, 1));         // [1, 2, 3]
console.log([1, 2, 3].fill(4, 3, 3));         // [1, 2, 3]
console.log([1, 2, 3].fill(4, -3, -2));       // [4, 2, 3]
console.log([1, 2, 3].fill(4, NaN, NaN));     // [1, 2, 3]
console.log([1, 2, 3].fill(4, 3, 5));         // [1, 2, 3]
console.log(Array(3).fill(4));                // [4, 4, 4]
console.log([].fill.call({ length: 3 }, 4));  // {0: 4, 1: 4, 2: 4, length: 3}

// Objects by reference.
var arr = Array(3).fill({}); // [{}, {}, {}]
arr[0].hi = "hi"; // [{ hi: "hi" }, { hi: "hi" }, { hi: "hi" }]
console.log(arr);
```

```
▶ (3) [4, 4, 4]
▶ (3) [1, 4, 4]
▶ (3) [1, 4, 3]
▶ (3) [1, 2, 3]
▶ (3) [1, 2, 3]
▶ (3) [4, 2, 3]
▶ (3) [1, 2, 3]
▶ (3) [1, 2, 3]
▶ (3) [4, 4, 4]
▶ {0: 4, 1: 4, 2: 4, length: 3}
▼ (3) [{…}, {…}, {…}] i
  ▶ 0: {hi: "hi"}
  ▶ 1: {hi: "hi"}
  ▶ 2: {hi: "hi"}
    length: 3
  ▶ __proto__: Array(0)
```

Array 확장

- unshift()
  - 새로운 요소를 배열의 맨 앞쪽에 추가하고, 새로운 길이를 반환한다.
  - 형식 : arr.unshift([...elementN])

매개변수명	설명
elementN	배열 맨 앞에 추가할 요소.

```
var arr = [1, 2];

console.log(arr.unshift(0)); // arr is [0, 1, 2]
console.log(arr);
console.log(arr.unshift(-2, -1)); // arr is [-2, -1, 0, 1, 2]
console.log(arr);
console.log(arr.unshift([-3]));// arr is [[-3], -2, -1, 0, 1, 2]
console.log(arr);
```

```
3
▶ (3) [0, 1, 2]
5
▶ (5) [-2, -1, 0, 1, 2]
6
▶ (6) [Array(1), -2, -1, 0, 1, 2]
```

Array 확장

- includes()
  - 배열이 특정 요소를 포함하고 있는지 판별한다
  - 형식 : arr.includes(valueToFind[, fromIndex])

매개변수명	설명
valueToFind	탐색할 요소.
fromIndex	이 배열에서 searchElement 검색을 시작할 위치. 음의 값은 array.length + fromIndex 의 인덱스를 asc로 검색한다. 기본값은 0

```
console.log([1, 2, 3].includes(2));     // true
console.log([1, 2, 3].includes(4));     // false
console.log([1, 2, 3].includes(3, 3));  // false
console.log([1, 2, 3].includes(3, -1)); // true
console.log([1, 2, NaN].includes(NaN)); // true

// fromIndex 가 배열의 길이보다 같거나 큰 경우 false를 반환한다.
var arr = ['a', 'b', 'c'];

console.log(arr.includes('c', 3));   // false
console.log(arr.includes('c', 100)); // false

// 0보다 작은 인덱스의 계산
// fromIndex 가 음수라면, valueToFind 를 찾기 시작할 배열의 위치로
// 사용되기 위해 연산된다. 만약 계산된 인덱스가 -1 * array.length
// 보다 작거나 같다면, 전체 배열이 검색된다.
var arr = ['a', 'b', 'c'];

console.log(arr.includes('a', -100)); // true
console.log(arr.includes('b', -100)); // true
console.log(arr.includes('c', -100)); // true
console.log(arr.includes('a', -2)); // false
```

Array 확장

- indexOf()
  - 배열에서 지정된 요소를 찾을 수 있는 첫 번째 인덱스를 반환하고 존재하지 않으면 -1을 반환한다.
  - 형식 : arr.indexOf(searchElement[, fromIndex])

매개변수명	설명
searchElement	배열에서 찾을 요소
fromIndex	검색을 시작할 색인. 인덱스가 배열의 길이보다 크거나 같은 경우 -1이 반환되므로 배열이 검색되지 않는다. 제공된 색인 값이 음수이면 배열 끝에서 부터의 오프셋 값으로 사용된다. 참고 : 제공된 색인이 음수이면 배열은 여전히 앞에서 뒤로 검색된다. 계산 된 인덱스가 0보다 작 으면 전체 배열이 검색된다. 기본값 : 0 (전체 배열 검색).

```
var array = [2, 9, 9];
console.log(array.indexOf(2));     // 0
console.log(array.indexOf(7));     // -1
console.log(array.indexOf(9, 2));  // 2
console.log(array.indexOf(2, -1)); // -1
console.log(array.indexOf(2, -3)); // 0

// 요소의 모든 항목 찾기
var indices = [];
var array = ['a', 'b', 'a', 'c', 'a', 'd'];
var element = 'a';
var idx = array.indexOf(element);
while (idx != -1) {
  indices.push(idx);
  idx = array.indexOf(element, idx + 1);
}
console.log(indices); // [0, 2, 4]
```

Array 확장

- lastIndexOf()
  - 지정된 요소가 배열에서 발견 될 수 있는 마지막 인덱스를 반환하고, 존재하지 않으면 -1을 반환한다. 배열은 fromIndex에서 시작하여 뒤로 검색된다.
  - 형식 : arr.lastIndexOf(searchElement[, fromIndex])

매개변수명	설명
searchElement	배열에서 찾을 요소
fromIndex	거꾸로 검색을 시작하는 인덱스이다. 배열의 길이에서 1을 뺀 값 (arr.length – 1)을 기본값으로 한다. 즉, 전체 배열이 검색된다. 인덱스가 배열의 길이보다 크거나 같으면 전체 배열이 검색된다. 부의 경우는, 배열의 마지막으로부터의 오프셋 (offset)로서 받아 들여진다. 인덱스가 음수 일지라도 배열은 계속해서 앞뒤로 검색된다. 계산 된 인덱스가 0보다 작은 경우 -1이 반환된다. 즉, 배열이 검색되지 않는다.

Array 확장

- lastIndexOf()

```
var array = [2, 5, 9, 2];
console.log(array.lastIndexOf(2));     // 3
console.log(array.lastIndexOf(7));     // -1
console.log(array.lastIndexOf(2, 3));  // 3
console.log(array.lastIndexOf(2, 2));  // 0
console.log(array.lastIndexOf(2, -2)); // 0
console.log(array.lastIndexOf(2, -1)); // 3

// 요소의 모든 항목 찾기
var indices = [];
var array = ['a', 'b', 'a', 'c', 'a', 'd'];
var element = 'a';
var idx = array.lastIndexOf(element);
while (idx != -1) {
  indices.push(idx);
  idx = (idx > 0 ? array.lastIndexOf(element, idx - 1) : -1);
}

console.log(indices);// [4, 2, 0]
```

```
3
-1
3
0
0
3
▶ (3) [4, 2, 0]
```

Array 확장

- toString()
  - 지정된 배열 및 그 요소를 나타내는 문자열을 반환한다.

```
var monthNames = ['Jan', 'Feb', 'Mar', 'Apr'];
var myVar = monthNames.toString(); // 'Jan,Feb,Mar,Apr'을 myVar에 할당.
console.log(myVar);
```

```
Jan,Feb,Mar,Apr
```

Array 확장

- entries()
  - 배열의 각 인덱스에 대한 키/값 쌍을 가지는 새로운 Array Iterator 객체를 반환한다.

```
var a = ['a', 'b', 'c'];
var iterator = a.entries();

for (let e of iterator) {
  console.log(e);
}
```

```
▶ (2) [0, "a"]
▶ (2) [1, "b"]
▶ (2) [2, "c"]
```

Array 확장

- every()
  - 배열 안의 모든 요소가 주어진 판별 함수를 통과하는지 테스트한다.
  - 형식 : arr.every(callback[, thisArg])
  - callback이 모든 배열 요소에 대해 참(truthy)인 값을 반환하는 경우 true, 그 외엔 false.

매개변수명	설명
callback	각 요소를 시험할 함수. 다음 세 가지 인수를 받는다. currentValue - 처리할 현재 요소. index (Optional) - 처리할 현재 요소의 인덱스. array (Optional) - every를 호출한 배열.
thisArg	callback을 실행할 때 this로 사용하는 값.

```
function isBigEnough(element, index, array) {
  return element >= 10;
}
console.log([12, 5, 8, 130, 44].every(isBigEnough));   // false
console.log([12, 54, 18, 130, 44].every(isBigEnough)); // true

console.log([12, 5, 8, 130, 44].every(elem => elem >= 10)); // false
console.log([12, 54, 18, 130, 44].every(elem => elem >= 10)); // true
```

Array 확장

- filter()
  - 주어진 함수의 테스트를 통과하는 모든 요소를 모아 새로운 배열로 반환한다.
  - 형식 : arr.filter(callback(element[, index[, array]])[, thisArg])
  - 테스트를 통과한 요소로 이루어진 새로운 배열. 어떤 요소도 테스트를 통과하지 못했으면 빈 배열을 반환된다.

매개변수명	설명
callback	각 요소를 시험할 함수. true를 반환하면 요소를 유지하고, false를 반환한다. 다음 세 가지 매개변수를 받는다. element - 처리할 현재 요소. index Optional - 처리할 현재 요소의 인덱스. array Optional - filter를 호출한 배열.
thisArg	callback을 실행할 때 this로 사용하는 값.

```
function isBigEnough(value) {
  return value >= 10;
}

var filtered = [12, 5, 8, 130, 44].filter(isBigEnough);
console.log(filtered) // [12, 130, 44]
```

Array 확장

- find()
  - 주어진 판별 함수를 만족하는 첫 번째 요소의 값을 반환합니다. 그런 요소가 없다면 undefined를 반환한다.
  - 형식 : arr.find(callback[, thisArg])
  - 주어진 판별 함수를 만족하는 첫 번째 요소의 값. 그 외에는 undefined.

매개변수명	설명
callback	배열의 각 값에 대해 실행할 함수. 아래의 세 인자를 받는다 element - 콜백함수에서 처리할 현재 요소. indexOptional - 콜백함수에서 처리할 현재 요소의 인덱스. arrayOptional - find 함수를 호출한 배열.
thisArg	선택 항목. 콜백이 호출될 때 this로 사용할 객체.

Array 확장

- find()

```
var inventory = [
    {name: 'apples', quantity: 2},
    {name: 'bananas', quantity: 0},
    {name: 'cherries', quantity: 5}
];

function findCherries(fruit) {
    return fruit.name === 'cherries';
}

console.log(inventory.find(findCherries)); // { name: 'cherries', quantity: 5 }
```

```
▶ {name: "cherries", quantity: 5}
```

Array 확장

- find()

```
const inventory = [
    {name: 'apples', quantity: 2},
    {name: 'bananas', quantity: 0},
    {name: 'cherries', quantity: 5}
];

const result = inventory.find(fruit => fruit.name === 'cherries');

console.log(result) // { name: 'cherries', quantity: 5 }
```

```
▶ {name: "cherries", quantity: 5}
```

```
// 배열에서 소수 찾기
function isPrime(element, index, array) {
  var start = 2;
  while (start <= Math.sqrt(element)) {
    if (element % start++ < 1) {
      return false;
    }
  }
  return element > 1;
}

console.log([4, 6, 8, 12].find(isPrime)); // undefined, not found
console.log([4, 5, 8, 12].find(isPrime)); // 5
```

Array 확장

- findIndex()
  - 주어진 판별 함수를 만족하는 배열의 첫 번째 요소에 대한 인덱스를 반환한다. 만족하는 요소가 없으면 -1을 반환한다.
  - 형식 : arr.findIndex(callback(element[, index[, array]])[, thisArg])
  - 요소가 테스트를 통과하면 배열의 인덱스. 그렇지 않으면 -1

매개변수명	설명
callback	3개의 인수를 취하여 배열의 각 값에 대해 실행할 함수이다. element - 배열에서 처리중인 현재 요소. index - 배열에서 처리중인 현재 요소의 인덱스. array - findIndex 함수가 호출된 배열.
thisArg	선택 항목. 콜백이 호출될 때 this로 사용할 객체.

Array 확장

- findIndex()

```
// 배열에서 소수 (소수가없는 경우 -1을 반환) 인 요소의 인덱스를 찾습니다.
function isPrime(element, index, array) {
  var start = 2;
  while (start <= Math.sqrt(element)) {
    if (element % start++ < 1) {
      return false;
    }
  }
  return element > 1;
}

console.log([4, 6, 8, 12].findIndex(isPrime)); // -1, not found
console.log([4, 6, 7, 12].findIndex(isPrime)); // 2
```

Array 확장

- forEach()
  - 주어진 함수를 배열 요소 각각에 대해 실행한다
  - 형식 : arr.forEach(callback(currentvalue[, index[, array]])[, thisArg]);

매개변수명	설명
callback	callback 각 요소에 대해 실행할 함수. 다음 세 가지 인수를 받는다. currentValue 처리할 현재 요소. index Optional 처리할 현재 요소의 인덱스. array Optional forEach()를 호출한 배열.
thisArg	선택 항목. 콜백이 호출될 때 this로 사용할 값.

Array 확장

- forEach()

```
// for 반복문을 forEach로 바꾸기
const items = ['item1', 'item2', 'item3'];
const copy = [];

// 이전
for (let i=0; i<items.length; i++) {
  copy.push(items[i]);
}
console.log(copy);

// 이후
items.forEach(function(item){
  copy.push(item);
});
console.log(copy);
```

```
▶ (3) ["item1", "item2", "item3"]
▶ (6) ["item1", "item2", "item3", "item1", "item2", "item3"]
```

Array 확장

- forEach()

```
// 다음 코드는 배열의 각 요소에 대해 한 줄을 기록합니다:
function logArrayElements(element, index, array) {
  console.log('a[' + index + '] = ' + element);
}

// 인덱스 2는 배열의 그 위치에 항목이 없기에
// 건너뜀을 주의하세요.
[2, 5, , 9].forEach(logArrayElements);
// 기록:
// a[0] = 2
// a[1] = 5
// a[3] = 9
```

```
// 배열의 각 항목에서 객체의 속성을 갱신합니다:
function Counter() {
  this.sum = 0;
  this.count = 0;
}
// thisArg 매개변수(this)를 forEach()에 제공했기에,
// callback은 전달받은 this의 값을 자신의 this 값으로 사용할 수 있습니다.
Counter.prototype.add = function(array) {
  array.forEach(function(entry) {
    this.sum += entry;
    ++this.count;
  }, this);
  // ^---- 주의
};

var obj = new Counter();
obj.add([2, 5, 9]);
console.log(obj.count); // 3
console.log(obj.sum); // 16
```

Array 확장

- keys()
  - 배열의 각 인덱스를 키 값으로 가지는 새로운 Array Iterator 객체를 반환한다.
  - 형식 : arr.keys();

```
const array1 = ['a', 'b', 'c'];
const iterator = array1.keys();

for (const key of iterator) {
  console.log(key);
}
```

```
0
1
2
```

Array 확장

- map()
  - 배열 내의 모든 요소 각각에 대하여 주어진 함수를 호출한 결과를 모아 새로운 배열을 반환한다.
  - 형식 : arr.map(callback(currentValue[, index[, array]])[, thisArg]);

매개변수명	설명
callback	새로운 배열 요소를 생성하는 함수. 다음 세 가지 인수를 가진다. currentValue - 처리할 현재 요소. index Optional - 처리할 현재 요소의 인덱스. array Optional - map()을 호출한 배열.
thisArg	callback을 실행할 때 this로 사용되는 값.

Array 확장

- map()

```
const array1 = [1, 4, 9, 16];

const map1 = array1.map(x => x * 2);
console.log(map1);

var numbers = [1, 4, 9];
var roots = numbers.map(Math.sqrt);
console.log(roots);
```

```
▶ (4) [2, 8, 18, 32]
▶ (3) [1, 2, 3]
```

Array 확장

- map()

```
// map을 활용해 배열 속 객체를 재구성하기
var kvArray = [{key:1, value:10},{key:2, value:20},{key:3, value: 30}];

var reformattedArray = kvArray.map(function(obj){
   var rObj = {};
   rObj[obj.key] = obj.value;
   return rObj;
});

console.log(reformattedArray); // [{1:10}, {2:20}, {3:30}]
console.log(kvArray); // [{key:1, value:10},{key:2, value:20},{key:3, value: 30}]

// 인자를 받는 함수를 사용하여 숫자 배열 재구성하기
var numbers = [1, 4, 9];
var doubles = numbers.map(function(num) {
    return num * 2;
});
console.log(doubles);

// map을 포괄적으로 사용하기
var map = Array.prototype.map;
var a = map.call('Hello World', function(x) { return x.charCodeAt(0); });
console.log(a); // [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100]
```

```
▼(3) [{…}, {…}, {…}]
  ▶0: {1: 10}
  ▶1: {2: 20}
  ▶2: {3: 30}
    length: 3
  ▶__proto__: Array(0)
▼(3) [{…}, {…}, {…}]
  ▶0: {key: 1, value: 10}
  ▶1: {key: 2, value: 20}
  ▶2: {key: 3, value: 30}
    length: 3
  ▶__proto__: Array(0)
▶(3) [2, 8, 18]
▶(11) [72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100]
```

Array 확장

- map()

```
// 아래 라인을 보시면...
['1', '2', '3'].map(parseInt);
// 결과를 [1, 2, 3] 으로 기대할 수 있습니다.
// 그러나 실제 결과는 [1, NaN, NaN] 입니다.

// parseInt 함수는 보통 하나의 인자만 사용하지만, 두 개를 받을 수 있습니다.
// 첫 번째 인자는 변환하고자 하는 표현이고 두 번째는 숫자로 변환할 때 사용할 진법입니다.
// Array.prototype.map은 콜백에 세 가지 인자를 전달합니다.
// 배열의 값, 값의 인덱스, 그리고 배열
// 세 번째 인자는 parseInt가 무시하지만 두 번째 인자는 아닙니다.

function returnInt(element) {
  return parseInt(element, 10);
}

console.log(['1', '2', '3'].map(returnInt)); // [1, 2, 3]
// 실제 결과가 예상한 대로 배열의 숫자와 같습니다.

// 위와 같지만 더 간단한 화살표 표현식
console.log(['1', '2', '3'].map(str => parseInt(str)));

// 더 간단하게 해결할 수 있는 방법
console.log(['1', '2', '3'].map(Number)); // [1, 2, 3]
// 그러나 `parseInt`와 달리 float이나 지수표현도 반환합니다.
console.log(['1.1', '2.2e2', '3e300'].map(Number)); // [1.1, 220, 3e+300]
```

```
▶ (3) [1, 2, 3]
▶ (3) [1, 2, 3]
▶ (3) [1, 2, 3]
▶ (3) [1.1, 220, 3e+300]
```

Array 확장

- reduce()
  - 배열의 각 요소에 대해 주어진 리듀서(reducer) 함수를 실행하고, 하나의 결과값을 반환한다.
  - 형식 : arr.reduce(callback[, initialValue]);

매개변수명	설명
callback	배열의 각 요소에 대해 실행할 함수. 다음 네 가지 인수를 받는다. accumulator - 누산기accmulator는 콜백의 반환값을 누적한다. 콜백의 이전 반환값 또는, 콜백의 첫 번째 호출이면서 initialValue를 제공한 경우에는 initialValue의 값이다. currentValue - 처리할 현재 요소. currentIndex Optional - 처리할 현재 요소의 인덱스. initialValue를 제공한 경우 0, 아니면 1부터 시작합니다. array Optional reduce()를 호출한 배열.
initialValue	callback의 최초 호출에서 첫 번째 인수에 제공하는 값. 초기값을 제공하지 않으면 배열의 첫 번째 요소를 사용한다. 빈 배열에서 초기값 없이 reduce()를 호출하면 오류가 발생한다.

Array 확장

- reduce()

```
const array1 = [1, 2, 3, 4];
const reducer = (accumulator, currentValue) => accumulator + currentValue;

// 1 + 2 + 3 + 4
console.log(array1.reduce(reducer)); // expected output: 10

// 5 + 1 + 2 + 3 + 4
console.log(array1.reduce(reducer, 5)); // expected output: 15
```

Array 확장

- reduce()
  - initialValue을 제공하지 않으면 출력 가능한 형식

```
var maxCallback = ( acc, cur ) => Math.max( acc.x, cur.x );
var maxCallback2 = ( max, cur ) => Math.max( max, cur );

// initialValue 없이 reduce()
console.log([ { x: 22 }, { x: 42 } ].reduce( maxCallback )); // 42
console.log([ { x: 22 }           ].reduce( maxCallback )); // { x: 22 }
// console.log([                     ].reduce( maxCallback )); // TypeError

// map/reduce로 개선 - 비었거나 더 큰 배열에서도 동작함
console.log([ { x: 22 }, { x: 42 } ].map( el => el.x )
                     .reduce( maxCallback2, -Infinity ));
```

```
42
▶ {x: 22}
42
```

Array 확장

- reduce() 작동 원리

```
[0, 1, 2, 3, 4].reduce(function(accumulator, currentValue, currentIndex, array) {
  return accumulator + currentValue;
});
```

callback	accumulator	currentValue	currentIndex	array	반환 값
1번째 호출	0	1	1	[0, 1, 2, 3, 4]	1
2번째 호출	1	2	2	[0, 1, 2, 3, 4]	3
3번째 호출	3	3	3	[0, 1, 2, 3, 4]	6
4번째 호출	6	4	4	[0, 1, 2, 3, 4]	10

- 화살표 함수를 이용해 들어갈 경우

```
[0, 1, 2, 3, 4].reduce( (prev, curr) => prev + curr );
```

Array 확장

- reduce() 작동 원리

```
[0, 1, 2, 3, 4].reduce(function(accumulator, currentValue, currentIndex, array) {
  return accumulator + currentValue;
}, 10);
```

	accumulator	currentValue	currentIndex	array	반환값
1번째 호출	10	0	0	[0, 1, 2, 3, 4]	10
2번째 호출	10	1	1	[0, 1, 2, 3, 4]	11
3번째 호출	11	2	2	[0, 1, 2, 3, 4]	13
4번째 호출	13	3	3	[0, 1, 2, 3, 4]	16
5번째 호출	16	4	4	[0, 1, 2, 3, 4]	20

Array 확장

- reduceRight()
  - reduce() 와 기능은 똑같지만 오른쪽부터 연산을 한다.
  - 형식 : arr.reduceRight(callback[, initialValue]);

매개변수명	설명
callback	배열의 각 요소에 대해 실행할 함수. 다음 네 가지 인수를 받는다. accumulator - 누산기accmulator는 콜백의 반환값을 누적한다. 콜백의 이전 반환값 또는, 콜백의 첫 번째 호출이면서 initialValue를 제공한 경우에는 initialValue의 값이다. currentValue - 처리할 현재 요소. currentIndex Optional - 처리할 현재 요소의 인덱스. initialValue를 제공한 경우 0, 아니면 1부터 시작합니다. array Optional reduce()를 호출한 배열.
initialValue	callback의 최초 호출에서 첫 번째 인수에 제공하는 값. 초기값을 제공하지 않으면 배열의 첫 번째 요소를 사용한다. 빈 배열에서 초기값 없이 reduce()를 호출하면 오류가 발생한다.

Array 확장

- reduceRight()

```
[0, 1, 2, 3, 4].reduceRight(function(previousValue, currentValue, index, array) {
    return previousValue + currentValue;
});
```

callback	accumulator	currentValue	currentIndex	array	반환 값
1번째 호출	4	3	3	[0, 1, 2, 3, 4]	7
2번째 호출	7	2	2	[0, 1, 2, 3, 4]	9
3번째 호출	9	1	1	[0, 1, 2, 3, 4]	10
4번째 호출	10	0	0	[0, 1, 2, 3, 4]	10

Array 확장

- reduceRight()

```
[0, 1, 2, 3, 4].reduceRight(function(previousValue, currentValue, index, array) {
    return previousValue + currentValue;
}, 10);
```

	accumulator	currentValue	currentIndex	array	반환값
1번째 호출	10	4	4	[0, 1, 2, 3, 4]	14
2번째 호출	14	3	3	[0, 1, 2, 3, 4]	17
3번째 호출	17	2	2	[0, 1, 2, 3, 4]	19
4번째 호출	19	1	1	[0, 1, 2, 3, 4]	20
5번째 호출	20	0	0	[0, 1, 2, 3, 4]	20

Array 확장

- reduceRight()

```
// 배열 내 모든 값의 합계 구하기
var sum = [0, 1, 2, 3].reduceRight(function(a, b) { return a + b; });
console.log(sum);  // 6

// 이중 배열 전개하기
var flattened = [[0, 1], [2, 3], [4, 5]].reduceRight(function(a, b) { return a.concat(b); }, []);
console.log(flattened);  // [4, 5, 2, 3, 0, 1]

// reduce와 reduceRight의 차이점
var a = ["1", "2", "3", "4", "5"];
var left  = a.reduce(function(prev, cur)      { return prev + cur; });
var right = a.reduceRight(function(prev, cur) { return prev + cur; });

console.log(left);  // "12345"
console.log(right); // "54321"
```

```
6
▶ (6) [4, 5, 2, 3, 0, 1]
12345
54321
```

Array 확장

- some()
  - 배열 안의 어떤 요소라도 주어진 판별 함수를 통과하는지 테스트한다.
  - 형식 : arr.some(callback[, thisArg]);

매개변수명	설명
callback	각 요소를 시험할 함수. 다음 세 가지 인수를 받는다. currentValue - 처리할 현재 요소. index Optional - 처리할 현재 요소의 인덱스. array Optional - some을 호출한 배열.
thisArg	callback을 실행할 때 this로 사용하는 값.

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

const even = (element) => element % 2 === 0;
console.log(array.some(even)); // true
```

Array 확장

- values()
  - 배열의 각 인덱스에 대한 값을 가지는 새로운 Array Iterator 객체를 반환한다.
  - 형식 : arr.values()

```
// for...of 루프를 통한 반복
var arr = ['w', 'y', 'k', 'o', 'p'];
var eArr = arr.values();
for (let letter of eArr) {
    console.log(letter);
}

// 다른 반복법
var arr = ['w', 'y', 'k', 'o', 'p'];
var eArr = arr.values();
console.log(eArr.next().value); // w
console.log(eArr.next().value); // y
console.log(eArr.next().value); // k
console.log(eArr.next().value); // o
console.log(eArr.next().value); // p
```