

배열 고차 함수

고차 함수(Higher order function)는 함수를 인자로 전달받거나 함수를 결과로 반환하는 함수를 말한다. 다시 말해, 고차 함수는 인자로 받은 함수를 필요한 시점에 호출하거나 클로저를 생성하여 반환한다. 자바스크립트의 함수는 일급 객체이므로 값처럼 인자로 전달할 수 있으며 반환할 수도 있다.

```
// 함수를 인자로 전달받고 함수를 반환하는 고차 함수
function makeCounter(predicate) {
  // 자유 변수. num의 상태는 유지되어야 한다.
  let num = 0;
  // 클로저. num의 상태를 유지한다.
  return function () {
    // predicate는 자유 변수 num의 상태를 변화시킨다.
    num = predicate(num);
    return num;
  };
}

// 보조 함수
function increase(n) {
  return ++n;
}

// 보조 함수
function decrease(n) {
  return --n;
}



// makeCounter는 함수를 인수로 전달받는다. 그리고 클로저를 반환한다.
const increaser = makeCounter(increase);
console.log(increaser()); // 1
console.log(increaser()); // 2
```

```
// makeCounter는 함수를 인수로 전달받는다. 그리고 클로저를 반환한다.  
const decreaser = makeCounter(decrease);  
console.log(decreaser()); // -1  
console.log(decreaser()); // -2
```

고차 함수는 외부 상태 변경이나 가변(mutable) 데이터를 피하고 **불변성(Immutability)**을 지향하는 함수형 프로그래밍에 기반을 두고 있다. 함수형 프로그래밍은 순수 함수(Pure function)와 보조 함수의 조합을 통해 로직 내에 존재하는 조건문과 반복문을 제거하여 복잡성을 해결하고 변수의 사용을 억제하여 상태 변경을 피하려는 프로그래밍 패러다임이다. 조건문이나 반복문은 로직의 흐름을 이해하기 어렵게 하여 가독성을 해치고, 변수의 값은 누군가에 의해 언제든지 변경될 수 있어 오류 발생의 근본적 원인이 될 수 있기 때문이다.

함수형 프로그래밍은 결국 순수 함수를 통해 **부수 효과(Side effect)**를 최대한 억제하여 오류를 피하고 프로그램의 안정성을 높이려는 노력의 한 방법이라고 할 수 있다.

자바스크립트는 고차 함수를 다수 지원하고 있다. 특히 Array 객체는 매우 유용한 고차 함수를 제공한다. 이들 함수에 대해 살펴보도록 하자.

-  메소드는 `this` (원본 배열)를 변경한다.
-  메소드는 `this` (원본 배열)를 변경하지 않는다.

Array.prototype.sort(compareFn?: (a: T, b: T) => number): this ES1

배열의 요소를 적절하게 정렬한다. 원본 배열을 직접 변경하며 정렬된 배열을 반환한다.

Array.prototype.sort 메서드는 10개 이상의 요소가 있는 배열을 정렬할 때 불안정한 알고리즘인 quicksort 알고리즘을 사용했다. 배열이 올바르게 정렬되도록 ECMAScript 2019는 Array.prototype.sort 메서드에 Timsort 알고리즘을 사용한다.

자세한 내용은 2019년과 이후 JavaScript의 동향 - JavaScript(ECMAScript)를 참고하기 바란다.

```
const fruits = ['Banana', 'Orange', 'Apple'];

// ascending(오름차순)
fruits.sort();
console.log(fruits); // [ 'Apple', 'Banana', 'Orange' ]

// descending(내림차순)
fruits.reverse();
console.log(fruits); // [ 'Orange', 'Banana', 'Apple' ]
```

주의할 것은 숫자를 정렬할 때이다. 아래 코드를 살펴보자.

```
const points = [40, 100, 1, 5, 2, 25, 10];

points.sort();
console.log(points); // [ 1, 10, 100, 2, 25, 40, 5 ]
```

기본 정렬 순서는 문자열 Unicode 코드 포인트 순서에 따른다. 배열의 요소가 숫자 타입이라 할지라도 배열의 요소를 일시적으로 문자열로 변환한 후, 정렬한다.

예를 들어, 문자열 '1'의 Unicode 코드 포인트는 `U+0031`, 문자열 '2'의 Unicode 코드 포인트는 `U+0032` 이다. 따라서 문자열 '1'의 Unicode 코드 포인트 순서가 문자열 '2'의 Unicode 코드 포인트 순서보다 앞서므로 문자열 '1'과 '2'를 sort 메소드로 정렬하면 1이 2보다 앞으로 정렬된다. 하지만 10의 Unicode 코드 포인트는 `U+0031U+0030` 이므로 2와 10를 sort 메소드로 정렬하면 10이 2보다 앞으로 정렬된다.

이러한 경우, sort 메소드의 인자로 정렬 순서를 정의하는 비교 함수를 인수로 전달한다. 비교 함수를 생략하면 배열의 각 요소는 일시적으로 문자열로 변환되어 Unicode 코드 포인트 순서에 따라 정렬된다.

```

const points = [40, 100, 1, 5, 2, 25, 10];

// 숫자 배열 오름차순 정렬
// 비교 함수의 반환값이 0보다 작은 경우, a를 우선하여 정렬한다.
points.sort(function (a, b) { return a - b; });
// ES6 화살표 함수
// points.sort((a, b) => a - b);
console.log(points); // [ 1, 2, 5, 10, 25, 40, 100 ]

// 숫자 배열에서 최소값 취득
console.log(points[0]); // 1

// 숫자 배열 내림차순 정렬
// 비교 함수의 반환값이 0보다 큰 경우, b를 우선하여 정렬한다.
points.sort(function (a, b) { return b - a; });
// ES6 화살표 함수
// points.sort((a, b) => b - a);
console.log(points); // [ 100, 40, 25, 10, 5, 2, 1 ]

// 숫자 배열에서 최대값 취득
console.log(points[0]); // 100

```

객체를 요소로 갖는 배열을 정렬하는 예제는 아래와 같다.

```

const todos = [
  { id: 4, content: 'JavaScript' },
  { id: 1, content: 'HTML' },
  { id: 2, content: 'CSS' }
];

// 비교 함수
function compare(key) {
  return function (a, b) {
    // 프로퍼티 값이 문자열인 경우, - 산술 연산으로 비교하면 NaN이 나오므로 비교 연산을 사용한다.
    return a[key] > b[key] ? 1 : (a[key] < b[key] ? -1 :

```

```

    0);
    };
}

// id를 기준으로 정렬
todos.sort(compare('id'));
console.log(todos);

// content를 기준으로 정렬
todos.sort(compare('content'));
console.log(todos);

```

Array.prototype.forEach(callback: (value: T, index: number, array: T[]) => void, thisArg?: any): void



ES5

- forEach 메소드는 for 문 대신 사용할 수 있다.
- 배열을 순회하며 배열의 각 요소에 대하여 인자로 주어진 콜백함수를 실행한다. **반환값은 undefined이다.**
- 콜백 함수의 매개변수를 통해 배열 요소의 값, 요소 인덱스, forEach 메소드를 호출한 배열, 즉 this를 전달 받을 수 있다.
- forEach 메소드는 원본 배열(this)을 변경하지 않는다. 하지만 콜백 함수는 원본 배열(this)을 변경할 수는 있다.
- **forEach 메소드는 for 문과는 달리 break 문을 사용할 수 없다.** 다시 말해, 배열의 모든 요소를 순회하며 중간에 순회를 중단할 수 없다.
- forEach 메소드는 for 문에 비해 성능이 좋지는 않다. 하지만 for 문보다 가독성이 좋으므로 적극 사용을 권장한다.
- IE 9 이상에서 정상 동작한다.

```

const numbers = [1, 2, 3];
let pows = [];

// for 문으로 순회

```

```

for (let i = 0; i < numbers.length; i++) {
  pows.push(numbers[i] ** 2);
}

console.log(pows); // [ 1, 4, 9 ]

pows = [];

// forEach 메소드로 순회
numbers.forEach(function (item) {
  pows.push(item ** 2);
});

// ES6 화살표 함수
// numbers.forEach(item => pows.push(item ** 2));

console.log(pows); // [ 1, 4, 9 ]

```

```

const numbers = [1, 3, 5, 7, 9];
let total = 0;

// forEach 메소드는 인수로 전달한 보조 함수를 호출하면서
// 3개(배열 요소의 값, 요소 인덱스, this)의 인수를 전달한다.
// 배열의 모든 요소를 순회하며 합산한다.
numbers.forEach(function (item, index, self) {
  console.log(`numbers[${index}] = ${item}`);
  total += item;
});

// Array#reduce를 사용해도 위와 동일한 결과를 얻을 수 있다
// total = numbers.reduce(function (pre, cur) {
//   return pre + cur;
// });

console.log(total); // 25
console.log(numbers); // [ 1, 3, 5, 7, 9 ]

```

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

// forEach 메소드는 원본 배열(this)을 변경하지 않는다. 하지만 콜백 함수는 원본 배열(this)을 변경할 수는 있다.

// 원본 배열을 직접 변경하려면 콜백 함수의 3번째 인자(this)를 사용한다.

```
numbers.forEach(function (item, index, self) {  
  self[index] = Math.pow(item, 2);  
});
```

```
console.log(numbers); // [ 1, 4, 9, 16 ]
```

// forEach 메소드는 for 문과는 달리 break 문을 사용할 수 없다.

```
[1, 2, 3].forEach(function (item, index, self) {  
  console.log(`self[${index}] = ${item}`);  
  if (item > 1) break; // SyntaxError: Illegal break statement  
});
```

forEach 메소드에 두번째 인자로 this를 전달할 수 있다.

```
function Square() {  
  this.array = [];  
}
```

```
Square.prototype.multiply = function (arr) {  
  arr.forEach(function (item) {  
    // this를 인수로 전달하지 않으면 this === window  
    this.array.push(item * item);  
  }, this);  
};
```

```
const square = new Square();  
square.multiply([1, 2, 3]);  
console.log(square.array); // [ 1, 4, 9 ]
```

ES6의 Arrow function를 사용하면 this를 생략하여도 동일한 동작을 한다.

```
Square.prototype.multiply = function (arr) {  
  arr.forEach(item => this.array.push(item * item));  
};
```

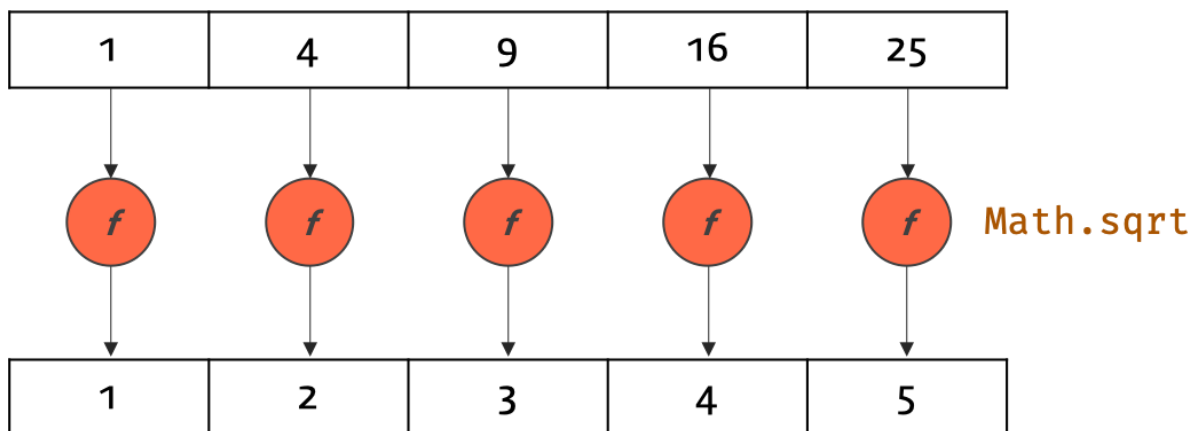
forEach 메소드의 이해를 돕기 위해 forEach의 동작을 흉내낸 myForEach 메소드를 작성해 보자.

```
Array.prototype.myForEach = function (f) {  
  // 첫번째 매개변수에 함수가 전달되었는지 확인  
  // console.log((function({})).toString()); // function()  
  {}  
  // console.log(Object.prototype.toString.call(function()  
  {})); // [object Function]  
  // poiemaweb.com/js-type-check 참고  
  if (!f || {}.toString.call(f) !== '[object Function]') {  
    throw new TypeError(f + ' is not a function.');  }  
  
  for (let i = 0; i < this.length; i++) {  
    // 배열 요소의 값, 요소 인덱스, forEach 메소드를 호출한 배열, 즉  
    this를 매개변수에 전달하고 콜백 함수 호출  
    f(this[i], i, this);  
  }  
};  
  
let total = 0;  
  
[0, 1, 2, 3].myForEach(function (item, index, array) {  
  console.log(`[${index}]: ${item} of [${array}]`);  
  total += item;  
});  
  
console.log('Total: ', total);
```


Array.prototype.map<U>(callbackfn: (value: T, index: number, array: T[]) => U, thisArg?: any): U[]

ES5

- 배열을 순회하며 각 요소에 대하여 인자로 주어진 콜백 함수의 반환값(결과값)으로 새로운 배열을 생성하여 반환한다. 이때 원본 배열은 변경되지 않는다.
- forEach 메소드는 배열을 순회하며 요소 값을 참조하여 무언가를 하기 위한 함수이며 map 메소드는 배열을 순회하며 요소 값을 다른 값으로 맵핑하기 위한 함수이다.



- 콜백 함수의 매개변수를 통해 배열 요소의 값, 요소 인덱스, map 메소드를 호출한 배열, 즉 this를 전달 받을 수 있다.

- IE 9 이상에서 정상 동작한다.

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

// 배열을 순회하며 각 요소에 대하여 인자로 주어진 콜백함수를 실행
const roots = numbers.map(function (item) {
  // 반환값이 새로운 배열의 요소가 된다. 반환값이 없으면 새로운 배열은
  // 비어 있다.
  return Math.sqrt(item);
});

// 위 코드의 축약표현은 아래와 같다.
// const roots = numbers.map(Math.sqrt);
```

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

map 메소드에 두번째 인자로 this를 전달할 수 있다.

```
function Prefixer(prefix) {
  this.prefix = prefix;
}

Prefixer.prototype.prefixArray = function (arr) {
  // 콜백함수의 인자로 배열 요소의 값, 요소 인덱스, map 메소드를 호출
  // 한 배열, 즉 this를 전달할 수 있다.
  return arr.map(function (x) {
    // x는 배열 요소의 값이다.
    return this.prefix + x; // 2번째 인자 this를 전달하지 않으면
    this === window
  }, this);
};

const pre = new Prefixer('-webkit-');
const preArr = pre.prefixArray(['linear-gradient', 'border-
radius']);
console.log(preArr);
// [ '-webkit-linear-gradient', '-webkit-border-radius' ]
```

ES6의 Arrow function를 사용하면 this를 생략하여도 동일한 동작을 한다.

map 메소드의 이해를 돕기 위해 map의 동작을 흉내낸 myMap 메소드를 작성해 보자.

```
Array.prototype.myMap = function (iteratee) {
  // 첫번째 매개변수에 함수가 전달되었는지 확인
  if (!iteratee || {}.toString.call(iteratee) !== '[object
  Function]') {
```

```

    throw new TypeError(iteratee + ' is not a function.');
```

```

  }

  const result = [];
  for (let i = 0, len = this.length; i < len; i++) {
    /**
     * 배열 요소의 값, 요소 인덱스, 메소드를 호출한 배열, 즉 this를
     매개변수를 통해 iteratee에 전달하고
     * iteratee를 호출하여 그 결과를 반환용 배열에 푸시하여 반환한다.
     */
    result.push(iteratee(this[i], i, this));
  }
  return result;
};

const result = [1, 4, 9].myMap(function (item, index, self)
{
  console.log(`[${index}]: ${item} of [${self}]`);
  return Math.sqrt(item);
});

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

Array.prototype.filter(callback: (value: T, index: number, array: Array) => any, thisArg?: any): T[]



ES5

- filter 메소드를 사용하면 if 문을 대체할 수 있다.
- 배열을 순회하며 각 요소에 대하여 인자로 주어진 콜백함수의 실행 결과가 true인 배열 요소의 값만을 추출한 새로운 배열을 반환한다.
- 배열에서 특정 케이스만 필터링 조건으로 추출하여 새로운 배열을 만들고 싶을 때 사용한다. 이때 원본 배열은 변경되지 않는다.
- 콜백 함수의 매개변수를 통해 배열 요소의 값, 요소 인덱스, filter 메소드를 호출한 배열, 즉 this를 전달 받을 수 있다.
- IE 9 이상에서 정상 동작한다.

```
const result = [1, 2, 3, 4, 5].filter(function (item, index, self) {
  console.log(`${index} = ${item}`);
  return item % 2; // 홀수만을 필터링한다 (1은 true로 평가된다)
});

console.log(result); // [ 1, 3, 5 ]
```

filter도 map, forEach와 같이 두번째 인자로 this를 전달할 수 있다.

filter 메소드의 이해를 돕기 위해 filter의 동작을 흉내낸 myFilter 메소드를 작성해 보자.

```
Array.prototype.myFilter = function (predicate) {
  // 첫번째 매개변수에 함수가 전달되었는지 확인
  if (!predicate || {}.toString.call(predicate) !== '[object Function]') {
    throw new TypeError(predicate + ' is not a function.');
```

}

```
const result = [];
for (let i = 0, len = this.length; i < len; i++) {
  /**
   * 배열 요소의 값, 요소 인덱스, 메소드를 호출한 배열, 즉 this를
   매개변수를 통해 predicate에 전달하고
   * predicate를 호출하여 그 결과가 참인 요소만을 반환용 배열에 푸
   시하여 반환한다.
   */
  if (predicate(this[i], i, this)) result.push(this[i]);
}
return result;
};

const result = [1, 2, 3, 4, 5].myFilter(function (item, index, self) {
  console.log(`${index}: ${item} of [${self}]`);
  return item % 2; // 홀수만을 필터링한다 (1은 true로 평가된다)
```

```
});
```

```
console.log(result); // [ 1, 3, 5 ]
```

Array.prototype.reduce<U>(callback: (state: U, element: T, index: number, array: T[]) => U, firstState?: U): U ES5

배열을 순회하며 각 요소에 대하여 이전의 콜백함수 실행 반환값을 전달하여 콜백함수를 실행하고 그 결과를 반환한다. IE 9 이상에서 정상 동작한다.

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

/*
previousValue: 이전 콜백의 반환값
currentValue : 배열 요소의 값
currentIndex : 인덱스
array        : 메소드를 호출한 배열, 즉 this
*/
// 합산
const sum = arr.reduce(function (previousValue, currentValue, currentIndex, self) {
  console.log(previousValue + '+' + currentValue + '=' + (previousValue + currentValue));
  return previousValue + currentValue; // 결과는 다음 콜백의 첫 번째 인자로 전달된다
});

console.log(sum); // 15: 1~5까지의 합
/*
1: 1+2=3
2: 3+3=6
3: 6+4=10
4: 10+5=15
15
*/
```

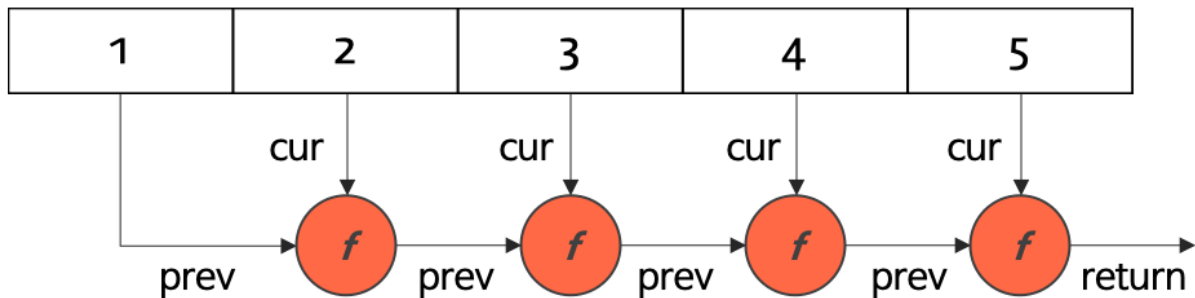
```

*/

// 최대값 취득
const max = arr.reduce(function (pre, cur) {
  return pre > cur ? pre : cur;
});

console.log(max); // 5: 최대값

```



Array.prototype.reduce의 두번째 인수로 초기값을 전달할 수 있다. 이 값은 콜백 함수에 최초로 전달된다.

```

const sum = [1, 2, 3, 4, 5].reduce(function (pre, cur) {
  return pre + cur;
}, 5);

console.log(sum); // 20
// 5 + 1 => 6 + 2 => 8 + 3 => 11 + 4 => 15 + 5

```

객체의 프로퍼티 값을 합산하는 경우를 생각해 보자.

```

const products = [
  { id: 1, price: 100 },
  { id: 2, price: 200 },
  { id: 3, price: 300 }
];

```

```
// 프로퍼티 값을 합산
const priceSum = products.reduce(function (pre, cur) {
  console.log(pre.price, cur.price);
  // 숫자값이 두번째 콜백 함수 호출의 인수로 전달된다. 이때 pre.price
  // 는 undefined이다.
  return pre.price + cur.price;
});

console.log(priceSum); // NaN
```

이처럼 객체의 프로퍼티 값을 합산하는 경우에는 반드시 초기값을 전달해야 한다.

```
const products = [
  { id: 1, price: 100 },
  { id: 2, price: 200 },
  { id: 3, price: 300 }
];

// 프로퍼티 값을 합산
const priceSum = products.reduce(function (pre, cur) {
  console.log(pre, cur.price);
  return pre + cur.price;
}, 0);

console.log(priceSum); // 600
```

reduce로 빈 배열을 호출하면 에러가 발생한다.

```
const sum = [].reduce(function (pre, cur) {
  console.log(pre, cur);
  return pre + cur;
});
// TypeError: Reduce of empty array with no initial value
```

초기값을 전달하면 에러를 회피할 수 있다. 따라서 reduce를 호출할 때는 **언제나 초기값을 전달하는 것이 보다 안전하다.**

```
const sum = [].reduce(function (pre, cur) {  
  console.log(pre, cur);  
  return pre + cur;  
}, 0);  
  
console.log(sum); // 0
```

Array.prototype.some(callback: (value: T, index: number, array: Array) => boolean, thisArg?: any): boolean ES5

배열 내 일부 요소가 콜백 함수의 테스트를 통과하는지 확인하여 그 결과를 boolean으로 반환한다. IE 9 이상에서 정상 동작한다.

콜백함수의 매개변수를 통해 배열 요소의 값, 요소 인덱스, 메소드를 호출한 배열, 즉 this를 전달 받을 수 있다.

```
// 배열 내 요소 중 10보다 큰 값이 1개 이상 존재하는지 확인  
let res = [2, 5, 8, 1, 4].some(function (item) {  
  return item > 10;  
});  
console.log(res); // false  
  
res = [12, 5, 8, 1, 4].some(function (item) {  
  return item > 10;  
});  
console.log(res); // true  
  
// 배열 내 요소 중 특정 값이 1개 이상 존재하는지 확인  
res = ['apple', 'banana', 'mango'].some(function (item) {  
  return item === 'banana';  
});
```



```
});  
console.log(res); // true
```

some()도 map(), forEach()와 같이 두번째 인자로 this를 전달할 수 있다.

Array.prototype.every(callback: (value: T, index: number, array: Array) => boolean, thisArg?: any): boolean ES5

배열 내 모든 요소가 콜백함수의 테스트를 통과하는지 확인하여 그 결과를 boolean으로 반환한다. IE 9 이상에서 정상 동작한다.

콜백함수의 매개변수를 통해 배열 요소의 값, 요소 인덱스, 메소드를 호출한 배열, 즉 this를 전달 받을 수 있다.

```
// 배열 내 모든 요소가 10보다 큰 값인지 확인  
let res = [21, 15, 89, 1, 44].every(function (item) {  
    return item > 10;  
});  
console.log(res); // false  
  
res = [21, 15, 89, 100, 44].every(function (item) {  
    return item > 10;  
});  
console.log(res); // true
```

every()도 map(), forEach()와 같이 두번째 인자로 this를 전달할 수 있다.

Array.prototype.find(predicate: (value: T, index: number, obj: T[]) => boolean, thisArg?: any): T | undefined ES6

ES6에서 새롭게 도입된 메소드로 Internet Explorer에서는 지원하지 않는다.

배열을 순회하며 각 요소에 대하여 인자로 주어진 콜백함수를 실행하여 그 결과가 참인 첫번째 요소를 반환한다. 콜백함수의 실행 결과가 참인 요소가 존재하지 않는다면 `undefined` 를 반환한다.

콜백함수의 매개변수를 통해 배열 요소의 값, 요소 인덱스, 메소드를 호출한 배열, 즉 `this`를 전달 받을 수 있다.

참고로 `filter`는 콜백함수의 실행 결과가 `true`인 배열 요소의 값만을 추출한 새로운 배열을 반환한다. 따라서 `filter`의 반환값은 언제나 배열이다. 하지만 `find`는 콜백함수를 실행하여 그 결과가 참인 첫번째 요소를 반환하므로 `find`의 결과값은 해당 요소값이다.

```
const users = [
  { id: 1, name: 'Lee' },
  { id: 2, name: 'Kim' },
  { id: 2, name: 'Choi' },
  { id: 3, name: 'Park' }
];

// 콜백함수를 실행하여 그 결과가 참인 첫번째 요소를 반환한다.
let result = users.find(function (item) {
  return item.id === 2;
});

// ES6
// const result = users.find(item => item.id === 2);

// Array#find는 배열이 아니라 요소를 반환한다.
console.log(result); // { id: 2, name: 'Kim' }

// Array#filter는 콜백함수의 실행 결과가 true인 배열 요소의 값만을
// 추출한 새로운 배열을 반환한다.
result = users.filter(function (item) {
  return item.id === 2;
});

console.log(result); // [ { id: 2, name: 'Kim' }, { id: 2, name: 'Choi' } ]
```

find 메소드의 이해를 돕기 위해 find의 동작을 흉내낸 myFind 메소드를 작성해 보자.

```
const users = [
  { id: 1, name: 'Lee' },
  { id: 2, name: 'Kim' },
  { id: 2, name: 'Choi' },
  { id: 3, name: 'Park' }
];

Array.prototype.myFind = function (predicate) {
  // 첫번째 매개변수에 함수가 전달되었는지 확인
  if (!predicate || {}.toString.call(predicate) !== '[object Function]') {
    throw new TypeError(predicate + ' is not a function.');
```

/**

- * 배열 요소의 값, 요소 인덱스, 메소드를 호출한 배열, 즉 this를 매개변수를 통해 predicate에 전달하고
- * predicate를 호출하여 그 결과가 참인 요소를 반환하고 처리를 종료한다.

*/

```
    for (let i = 0, len = this.length; i < len; i++) {
      if (predicate(this[i], i, this)) return this[i];
    }
  };

const result = users.myFind(function (item, index, array) {
  console.log(`[${index}]: ${JSON.stringify(item)} of [${JSON.stringify(array)}]`);
  return item.id === 2; // 요소의 id 프로퍼티의 값이 2인 요소를 검색
});

console.log(result); // { id: 2, name: 'Kim' }
```

Array.prototype.findIndex(predicate: (value: T, index: number, obj: T[]) => boolean, thisArg?: any): number ES6

ES6에서 새롭게 도입된 메소드로 Internet Explorer에서는 지원하지 않는다.

배열을 순회하며 각 요소에 대하여 인자로 주어진 콜백함수를 실행하여 그 결과가 참인 첫 번째 요소의 인덱스를 반환한다. 콜백함수의 실행 결과가 참인 요소가 존재하지 않는다면 -1을 반환한다.

콜백함수의 매개변수를 통해 배열 요소의 값, 요소 인덱스, 메소드를 호출한 배열, 즉 this를 전달 받을 수 있다.

```
const users = [
  { id: 1, name: 'Lee' },
  { id: 2, name: 'Kim' },
  { id: 2, name: 'Choi' },
  { id: 3, name: 'Park' }
];

// 콜백함수를 실행하여 그 결과가 참인 첫 번째 요소의 인덱스를 반환한다.
function predicate(key, value) {
  return function (item) {
    return item[key] === value;
  };
}

// id가 2인 요소의 인덱스
let index = users.findIndex(predicate('id', 2));
console.log(index); // 1

// name이 'Park'인 요소의 인덱스
index = users.findIndex(predicate('name', 'Park'));
console.log(index); // 3
```