9주차 JavaScript

상태	승인
설명	배열

배열

개발을 진행하다 보면 첫 번째 요소, 두 번째 요소, 세 번째 요소 등과 같이 *순서가 있는 컬렉션*이 필요할 때가 있다.

사용자나 물건 목록같이 일목요연하게 순서를 만들어 정렬하는 경우가 생기기 때문이다.

이런 경우에 사용하기 좋은 구조가 `배열`이다.

배열 선언

아래 두 문법을 사용하면 빈 배열을 만들 수 있다.

```
let arr = new Array();
let arr = [];
```

대부분 두 번째 방법으로 배열을 선언하,며 이때 대괄호 안에 초기 요소를 넣어주는 것도 가능하다.

```
let fruits = ["사과", "오렌지", "자두"];
```

각 배열 요소엔 0부터 시작하는 `숫자(인덱스)`가 매겨져 있는데, 이 숫자들은 배열 내 순서를 나타내게 된다.

배열 내 특정 요소를 얻고 싶다면 대괄호 안에 순서를 나타내는 숫자인 인덱스를 넣어주면 된다.

💡 언어에서의 인덱스 ❓

보통 우리는 숫자를 셀때 1부터 센다. 사과의 개수를 센다면 1개,2개,3개 이런식으로 말이다. 하지만 모든 컴퓨터언어는 숫자를 세는 경우 0부터 센다. 즉, 순서를 고른다면 0번째, 1번째, 2번째 이런 식으로 숫자를 세개 된다.

```
let fruits = ["사과", "오렌지", "자두"];

alert( fruits[0] ); // 사과
alert( fruits[1] ); // 오렌지
alert( fruits[2] ); // 자두
```

아래 방법으로 요소를 수정할 수 있다.

```
fruits[2] = '배'; // 배열이 ["사과", "오렌지", "배"]로 바뀜
alert(fruits);
```

새로운 요소를 배열에 추가하는 것도 가능하다.

```
fruits[3] = '레몬'; // 배열이 ["사과", "오렌지", "배", "레몬"]으로 바뀜
alert(fruits)
```

`length`를 사용하면 배열에 담긴 요소가 몇 개인지 알 수 있다.

```
let fruits = ["사과", "오렌지", "자두"];

alert( fruits.length ); // 3
```

배열 요소의 자료형엔 제약이 없으며 어떤 자료형이던지 섞여서 생성 혹은 수정이 가능하다.

```
// 요소에 여러 가지 자료형이 섞여 있습니다.
let arr = [ '사과', { name: '이보라' }, true, function() { alert('안녕하세요.'); } ];

// 인덱스가 1인 요소(객체)의 name 프로퍼티를 출력합니다.
alert( arr[1].name ); // 이보라

// 인덱스가 3인 요소(함수)를 실행합니다.
arr[3](); // 안녕하세요.
```

pop

배열 끝 요소를 제거하고, 제거한 요소를 반환한다.

```
let fruits = ["사과", "오렌지", "배"];

alert( fruits.pop() ); // 배열에서 "배"를 제거하고 제거된 요소를 얼럿창에 띄웁니다.

alert( fruits ); // 사과,오렌지
```

push

배열 끝에 요소를 추가한다.

```
let fruits = ["사과", "오렌지"];

fruits.push("배");

alert( fruits ); // 사과,오렌지,배

//fruits.push(...)를 호출하는 것은 fruits[fruits.length] = ...하는 것과 같은 동작을 한다.
```

shift

배열 앞 요소를 제거하고, 제거한 요소를 반환한다.

```
let fruits = ["사과", "오렌지", "배"];

alert( fruits.shift() ); // 배열에서 "사과"를 제거하고 제거된 요소를 얼럿창에 띄웁니다.

alert( fruits ); // 오렌지,배
```

unshift

배열 앞에 요소를 추가한다.

```
let fruits = ["오렌지", "배"];

fruits.unshift('사과');

alert( fruits ); // 사과,오렌지,배
```

`push`와 `unshift`는 요소 여러 개를 한 번에 더해줄 수도 있다.

```
let fruits = ["사과"];

fruits.push("오렌지", "배");
fruits.unshift("파인애플", "레몬");

// ["파인애플", "레몬", "사과", "오렌지", "배"]
alert( fruits );
```

반복문

`for`문은 배열을 순회할 때 쓰는 가장 오래된 방법이다. 순회시엔 인덱스를 사용한다.

```
let arr = ["사과", "오렌지", "배"];

for (let i = 0; i < arr.length; i++) {alert( arr[i] );
}
```

배열에 적용할 수 있는 또 다른 순회 문법으로론 `for..of`가 있다.

```
let fruits = ["사과", "오렌지", "자두"];

// 배열 요소를 대상으로 반복 작업을 수행합니다.
```

```
for (let fruit of fruits) {
  alert( fruit );
}
```

`for..of`를 사용하면 현재 요소의 인덱스는 얻을 수 없고 값만 얻을 수 있다. 이 정도 기능이면 원하는 것을 충분히 구현할 수 있고 문법도 짧기 때문에 배열의 요소를 대상으로 반복 작업을 할 땐 `for..of`를 사용해 보는 것을 추천한다.

length

배열에 무언가 조작을 가하면 `length` 프로퍼티가 자동으로 갱신된다. 엄밀히 말하면`length` 프로퍼티는 배열 내 요소의 개수가 아니라 가장 큰 인덱스에 1을 더한 값을 반환한다.

따라서 배열에 요소가 하나 있고, 이 요소의 인덱스가 아주 큰 정수라면 배열의 `length` 프로퍼티도 아주 커지게 된다.

```
let fruits = [];
fruits[123] = "사과";

alert( fruits.length ); // 124
```

하지만 물론 위와같은 방법으로 배열을 사용하는것은 옳지 않기에 사용하는 것은 권장되지 않는다.

다차원 배열

배열 역시 배열의 요소가 될 수 있으며 이러한 배열들을 다차원 배열이라고 한다.

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

alert( matrix[1][1] ); // 5, 중심에 있는 요소
```

다차원 배열이 잘 이해되지 않는다면 아래와 같이 생각해보자.

`matrix[1]` 은 `[1,2,3]`을 나타낸다는 것은 이해할 수 있을 것이다. 그렇다면 `matrix[1]` 자체를 하나의 변수라고 생각을 해보자.

```
const MAXTRIX = maxtrix[1]
```

위처럼 저장을 해둔다면 `MAXTRIX[1]` 은 `[1,2,3]`의 인덱스 `1`값이므로 `2`가 나올 것이다.

toString

배열엔 `toString` 메서드가 구현되어 있어 이를 호출하면 요소를 쉼표로 구분한 문자열이 반환된다.

```
let arr = [1, 2, 3];

alert( arr ); // 1,2,3
alert( String(arr) === '1,2,3' ); // true
```

요약

배열은 특수한 형태의 객체로, 순서가 있는 자료를 저장하고 관리하는 용도에 최적화된 자료구조이다.

1️⃣ 선언 방법:

```
// 대괄호 (가장 많이 쓰이는 방법임)
let arr = [item1, item2...];
```

2️⃣ `length` 프로퍼티는 배열의 길이를 나타내주며 정확히는 숫자형 인덱스 중 가장 큰 값에 1을 더한 값이다.

3️⃣ `push(...items)` – `items`를 배열 끝에 더해준다.

4️⃣ `pop()` – 배열 끝 요소를 제거하고, 제거한 요소를 반환한다.

5️⃣ `shift()` – 배열 처음 요소를 제거하고, 제거한 요소를 반환한다.

6️⃣ `unshift(...items)` – `items`를 배열 처음에 더해준다.

아래 방법을 사용하면 모든 요소를 대상으로 반복 작업을 할 수 있다.

7️⃣ `for (let i=0; i<arr.length; i++)` – 가장 빠른 방법이다.

8️⃣ `for (let item of arr)` – 배열 요소에만 사용되는 모던한 문법이다.

💡 문제 1

아래 코드를 실행하면 어떤 결과가 나올까요?

```
let fruits = ["사과", "배", "오렌지"];

// 배열을 '복사'한 후, push 메서드를 이용해 새로운 값을 추가합니다.
let shoppingCart = fruits;
shoppingCart.push("바나나");

// fruits에 어떤 값이 들어 있을까요?
alert( fruits.length ); // ?
```

💡 문제 2

배열과 관련된 다섯 가지 연산을 해봅시다.

1. 요소 “Jazz”, "Blues"가 있는 `styles` 배열을 생성합니다.
2. "Rock-n-Roll"을 배열 끝에 추가합니다.
3. 배열 정 중앙에 있는 요소를 "Classics"로 바꿉니다. 가운데 요소를 찾는 코드는 요소가 홀수 개인 배열에서도 잘 작동해야 합니다.
4. 배열의 첫 번째 요소를 꺼내서 출력합니다.
5. "Rap"과 "Reggae"를 배열의 앞에 추가합니다.

단계를 하나씩 거칠 때마다 배열 모습은 아래와 같이 변해야 합니다.

```
Jazz, Blues
Jazz, Blues, Rock-n-Roll
Jazz, Classics, Rock-n-Roll
Classics, Rock-n-Roll
Rap, Reggae, Classics, Rock-n-Roll
```

배열과 메서드

배열을 자유자재로 다룰 수 있는 능력은 프로그래밍에서 아주아주 중요한 능력 중 하나이다. 반복문을 활용하는것으로도 배열을 다룰 수 있지만, 배열을 더욱 더 쉽게 다룰 수 있고 자주 사용되는 내부 `메서드(함수)`들을 활용해보자.

delete

배열에서 요소를 하나만 지우고 싶다면 delete를 활용할 수 있다.

```
let arr = ["I", "go", "home"];

delete arr[1]; // "go"를 삭제합니다.

alert( arr[1] ); // undefined
```

```
// delete를 써서 요소를 지우고 난 후 배열 --> arr = ["I",  , "home"];
alert( arr.length ); // 3
```

원하는 대로 요소를 지웠지만 이상하게 배열의 길이를 확인하면 그대로 3개이다.

```
arr.length == 3
```

이는 자연스러운 결과로, delete는 해당 인덱스의 해당되는 값을 지우고 빈공간을 만들고 끝나기 때문에 배열의 공간자체를 없앨수는 없다.

Splice

위 경우 splice를 활용하면 해결할 수 있다.

```
arr.splice(index[, deleteCount, elem1, ..., elemN])
```

첫번째 인자로는 인덱스(index)를, 두번째 인자로는 인덱스붜 제거하고자 하는 요소의 개수를 넣어준다.

그 이후부터는 배열에 추가할 요소를 넣어준다.

말로 이해하기는 어려울 수 있으니 실습을 해보면서 이해해보자.

```
let arr = ["I", "study", "JavaScript"];

arr.splice(1, 1); // 인덱스 1부터 요소 한 개를 제거alert( arr ); // ["I", "JavaScript"]
```

인덱스 `1`이 가리키는 요소부터 시작해 요소 한 개(`1`)를 지우는 동작을 하게 된다.

```
let arr = ["I", "study", "JavaScript", "right", "now"];

// 처음(0) 세 개(3)의 요소를 지우고, 이 자리를 다른 요소로 대체합니다.
arr.splice(0, 3, "Let's", "dance");

alert( arr ) // now ["Let's", "dance", "right", "now"]
```

`splice`는 삭제된 요소로 구성된 배열을 반환한다. 즉, 자른 배열을 새로운 변수에 넣어줄 수 있다.

```
let arr = ["I", "study", "JavaScript", "right", "now"];

// 처음 두 개의 요소를 삭제함
let removed = arr.splice(0, 2);

alert( removed ); // "I", "study" <-- 삭제된 요소로 구성된 배열
```

`splice` 메서드의 `deleteCount`를 `0`으로 설정하면 요소를 제거하지 않으면서 새로운 요소를 추가하는 용도로도 사용이 가능하다.

```
let arr = ["I", "study", "JavaScript"];

// 인덱스 2부터
// 0개의 요소를 삭제합니다.
// 그 후, "complex"와 "language"를 추가합니다.
arr.splice(2, 0, "complex", "language");

alert( arr ); // "I", "study", "complex", "language", "JavaScript"
```

음수 인덱스도 사용할 수 있다.

slice 메서드 뿐만 아니라 배열 관련 `메서드(함수)`엔 음수 인덱스를 사용할 수 있다. 인덱스 `-1`은 배열 끝에서부터의 첫번째 요소를 의미한다.

```
let arr = [1, 2, 5];

// 인덱스 -1부터 (배열 끝에서부터 첫 번째 요소)
// 0개의 요소를 삭제하고
// 3과 4를 추가합니다.
arr.splice(-1, 0, 3, 4);

alert( arr ); // 1,2,3,4,5
```

Slice

`slice`는 `arr.splice`와 유사하며 사용방법이 조금 더 간단하다.

```
arr.slice([start], [end])
```

`splice`와는 조금 다르게 두개의 인자를 가지게 되며, 시작 인덱스부터 `end`인덱스까지의 요소를 복사해서 반환한다.

```
let arr = ["t", "e", "s", "t"];

alert( arr.slice(1, 3) ); // e,s (인덱스가 1인 요소부터 인덱스가 3인 요소까지를 복사(인덱스가 3인 요소는 제외))

alert( arr.slice(-2) ); // s,t (인덱스가 -2인 요소부터 제일 끝 요소까지를 복사)
```

💡 splice vs slice ❓

`splice`와 `slice`는 문법도 약간 다르지만 큰 차이가 하나 있다. `splice`는 원본 배열을 건드리면서 자르는 반면, `slice`는 원본배열이 유지되면서 배열을 자른다. 상황에 따라 맞는 함수를 사용하는것이 필요한 이유이다.

concat

`concat`은 기존 배열의 요소를 사용해 새로운 배열을 만들거나 기존 배열에 요소를 추가하고자 할 때 사용한다.

```
arr.concat(arg1, arg2...)
```

인수엔 배열이나 값이 올 수 있으며 개수에는 제한이 없다.

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

// arr의 요소 모두와 [3,4]의 요소 모두를 한데 모은 새로운 배열이 만들어집니다.
alert( arr.concat([3, 4]) ); // 1,2,3,4

// arr의 요소 모두와 [3,4]의 요소 모두, [5,6]의 요소 모두를 모은 새로운 배열이 만들어집니다.
alert( arr.concat([3, 4], [5, 6]) ); // 1,2,3,4,5,6

// arr의 요소 모두와 [3,4]의 요소 모두, 5와 6을 한데 모은 새로운 배열이 만들어집니다.
alert( arr.concat([3, 4], 5, 6) ); // 1,2,3,4,5,6
```

`concat` 메서드는 제공받은 배열의 요소를 복사해 활용한다. (`slice`와 유사)

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

let arrayLike = {
  0: "something",
  length: 1
};

alert( arr.concat(arrayLike) ); // 1,2,[object Object]
```

forEach로 반복작업 하기

`forEach`를 활용하면 주어진 함수를 배열 요소 각각에 대해 실행할 수 있다. 이전에 배운 `for of` 구문과 비슷하지만 조금 더 간결하게 사용할 수 있다.

```
arr.forEach(function(item, index, array) {
  // 요소에 무언가를 할 수 있습니다.
});
```

아래는 요소 모두를 `alert()`를통해 출력해주는 코드이다.

```
// for each element call alert
["Bilbo", "Gandalf", "Nazgul"].forEach(el => alert(el));
```

아래는 인덱스 정보까지 더해서 출력해주는 좀 더 정교한 코드이다.

```
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
  alert(`${item} is at index ${index} in ${array}`);
});
```

배열 탐색하기

배열 내에서 무언가를 찾고 싶을 때 쓰는 메서드에 대해 알아보자.

indexOf, lastIndexOf , includes

위 메서드는 같은 이름을 가진 문자열 메서드와 문법이 동일하며 물론 하는 일도 같다. 연산 대상이 문자열이 아닌 배열의 요소라는 점만 다르다.

- `arr.indexOf(item, from)`는 인덱스 `from`부터 시작해 `item(요소)`을 찾고 요소를 발견하면 해당 요소의 인덱스를 반환하고, 발견하지 못했으면 `-1`을 반환한다.
- `arr.lastIndexOf(item, from)`는 위 메서드와 동일한 기능을 하는데, 검색을 끝에서부터 시작한다는 점만 다르다.
- `arr.includes(item, from)`는 인덱스 `from`부터 시작해 `item`이 있는지를 검색하는데, 해당하는 요소를 발견하면 `true`를 반환한다.

```
let arr = [1, 0, false];

alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2
alert( arr.indexOf(null) ); // -1
```

```
alert( arr.includes(1) ); // true
```

find , findIndex

객체로 이루어진 배열이 있다고 가정해 보자. 특정 조건에 부합하는 객체를 배열 내에서 찾을때 위 메서드들을 활용해볼 수 있다.

```
let result = arr.find(function(item, index, array) {
  // true가 반환되면 반복이 멈추고 해당 요소를 반환합니다.
  // 조건에 해당하는 요소가 없으면 undefined를 반환합니다.
});
```

- `item` – 함수를 호출할 요소
- `index` – 요소의 인덱스
- `array` – 배열 자기 자신

함수가 참을 반환하면 탐색은 중단되고 해당 `요소`가 반환된다. 원하는 요소를 찾지 못했으면 `undefined`가 반환되게 된다.

배열 내에서 Pete인 사람을 찾아내는 예시를 한번 들어보자.

```
let users = [
  "John",
  "Pete",
  "Mary"
];

let user = users.find(item => item == "Pete");

alert(user); // Pete
```

filter

`find` 메서드는 함수의 반환 값을 `true`로 만드는 단 하나의 요소를 찾는데, 여러개를 찾고싶을때는 `filter`를 사용하면 된다.

`filter`는 `find`와 문법이 유사하며 조건을 충족하는 요소들을 배열에 담아서 반환해준다.

```
let results = arr.filter(function(item, index, array) {
  // 조건을 충족하는 요소는 results에 순차적으로 더해집니다.
  // 조건을 충족하는 요소가 하나도 없으면 빈 배열이 반환됩니다.
});
```

```
let users = [
  "John",
  "Pete",
  "Mary"
];

// 앞쪽 사용자 두 명을 반환합니다.
let someUsers = users.filter(item => item ==="John" || item ==="Pete");

alert(someUsers.length); // 2
```

배열을 변형시키는 메서드

배열을 변형시키거나 요소를 재 정렬해주는 메서드에 대해 알아보자.

map

`map`은 배열 요소 전체를 대상으로 함수를 호출하고, 함수 호출 결과를 배열로 반환해주며 굉장히 자주 사용되는 메소드 중 하나이다.

```
let result = arr.map(function(item, index, array) {
  // 요소 대신 새로운 값을 반환합니다.
});
```

아래 예시에선 각 요소의 길이를 배열로 가져온 후, 이를 출력해준다.

```
let lengths = ["Bilbo", "Gandalf", "Nazgul"].map(item => item.length);
// lengths => [5,7,6]
alert(lengths); // 5,7,6
```

sort

`sort()`는 배열의 요소를 정렬해주며, 배열의 원본 자체를 바꾼다.

```
let arr = [ 1, 2, 15 ];

// arr 내부가 재 정렬됩니다.
arr.sort();

alert( arr );  // 1, 15, 2
```

뭔가 나온 결과가 이상할 것이다.

재정렬 후 배열 요소가 `1, 15, 2`가 되었는데 기대하던 결과(`1, 2, 15`)와는 조금 다르게 나온다. 해당이유는 정렬의 요소는 모두 문자열로 취급되기 때문이다.

문자열 비교 ❓

자바스크립트에서 문자열을 비교하는 경우 `("2" < "15")` 아스키코드 순서로 편집되기 때문에 `2`가 `15`보다 크게 취급되게 된다.

아스키 코드 ❓

컴퓨터는 문자를 기억할 수 없다. 0과 1로만 구성된 숫자로 기억할 수 있다. 따라서 컴퓨터에게 문자를 기억하게 하기위한 일종의 사전이다.

제어 문자		공백 문자		구두점		숫자		알파벳			
10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	`
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x02	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x04	EOT	36	0x24	$	68	0x44	D	100	0x64	d
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g
8	0x08	BS	40	0x28	(	72	0x48	H	104	0x68	h
9	0x09	HT	41	0x29	)	73	0x49	I	105	0x69	i
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0x0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	ESC	59	0x3B	;	91	0x5B	[	123	0x7B	{
28	0x1C	FS	60	0x3C	<	92	0x5C	₩	124	0x7C	|
29	0x1D	GS	61	0x3D	=	93	0x5D	]	125	0x7D	}
30	0x1E	RS	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	US	63	0x3F	?	95	0x5F	_	127	0x7F	DEL

물론 사전 내용을 외울 필요는 없지만 자바스크립트에서 정렬이 들어가면 위 순서대로 정렬이 된다는 것을 알아두자.

우리가 원하는 기댓값으로 배열을 정렬하기 위해서는 sort() 함수에 인자로 또다른 함수를 넣어줘야 한다.

```
let arr = [1,15,2]

arr.sort((a,b) => a-b)
alert(arr); // 1, 2, 15
```

내림차순 정렬을 하려면 아래와 같이 하면 된다.

```
let arr = [1,15,2]

arr.sort((a,b) => b-a)
alert(arr); // 15 , 2, 1
```

아직 함수라는 개념을 배우지 않았고 위와같은 내용은 다른 프로그래밍 언어에서는 보기 힘든 속성이기때문에 우선은 자세히 알아보지 않고 넘어가자. 다만 정렬을 해야하는 경우가 온다면, 위와 같은 방식으로 코드를 작성하자.

reverse

`arr`의 요소를 역순으로 정렬시켜주는 메서드이다.

```
let arr = [1, 2, 3, 4, 5];
arr.reverse();

alert( arr ); // 5,4,3,2,1
```

split , join

split을 활용하면 split안의 인자로 들어가는 구분자를 활용해서 문자열을 해당 구분자를 기준으로 나눌 수 있다.

```
let names = 'Bilbo, Gandalf, Nazgul';

let arr = names.split(', ');

for (let name of arr) {
  alert( `${name}에게 보내는 메시지` ); // Bilbo에게 보내는 메시지
}
```

문자열을 글자 단위로 분리하기

`split(s)`의 `s`를 빈 문자열로 지정하면 문자열을 글자 단위로 분리할 수 있게 된다.

```
let str = "test";

alert( str.split('') ); // t,e,s,t
```

`join`은 `split`과 반대 역할을 하는 메서드이며. 인수 를 접착제처럼 사용해 배열 요소를 모두 합친 후 하나의 문자열을 만들어 준다.

```
let arr = ['Bilbo', 'Gandalf', 'Nazgul'];

let str = arr.join(';'); // 배열 요소 모두를 ;를 사용해 하나의 문자열로 합칩니다.

alert( str ); // Bilbo;Gandalf;Nazgul
```

ruduce

`forEach`, `for`, `for..of`를 사용하여 배열 내 요소를 대상으로 반복 작업을 할 수 있었다.

각 요소를 돌면서 반복 작업을 수행하고, 작업 결과물을 새로운 배열 형태로 얻으려면 `map`을 사용할 수 있었다.

`reduce`와 `reduceRight` 도 이런 메서드들과 유사한 작업을 해준다.

그런데 사용법이 조금 복잡하니 차근차근 따라해 보도록 하자. `reduce`와 `reduceRight`는 배열을 기반으로 값 하나를 도출할 때 사용된다.

```
let value = arr.reduce(function(accumulator, item, index, array) {
  // ...
}, [initial]);
```

함수의 인자가 4개나 되는데, 각각의 의미를 한번 알아보자.

- `accumulator` – 이전 함수 호출의 결과. `initial`은 함수 최초 호출 시 사용되는 초깃값을 나타냄(옵션)
- `item` – 현재 배열 요소
- `index` – 요소의 위치
- `array` – 배열

이전 함수 호출 결과는 다음 함수를 호출할 때 첫 번째 인수(`previousValue`)로 사용된다.

첫 번째 인수는 앞서 호출했던 함수들의 결과가 누적되어 저장되는 변수라 생각하자. 마지막 함수까지 호출되면`reduce`의 반환 값이 된다.

복잡해 보이긴 하지만 예제를 통해 메서드를 이해해 보자.

`reduce`를 이용해 코드 한 줄로 배열의 모든 요소를 더한 값을 구할 수 있다.

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

let result = arr.reduce((sum, current) => sum + current, 0);

alert(result); // 15
```

`reduce`에 전달한 함수는 오직 인수 두 개만 받고 있는데, 보통은 index와 array는 잘 사용되지 않는다.

1. 함수 최초 호출 시, `reduce`의 마지막 인수인 `0(초깃값)`이 `sum`에 할당된다. `current`엔 배열의 첫 번째 요소인 `1`이 할당되고 따라서 함수의 결과는 `1`이 된다.
2. 두 번째 호출 시, `sum = 1` 이고 여기에 배열의 두 번째 요소(`2`)가 더해지므로 결과는 `3`이 된다.
3. 세 번째 호출 시, `sum = 3` 이고 여기에 배열의 다음 요소가 더해지며 이런 과정이 계속 이어지게 된다.

	`sum`	`current`	result
첫 번째 호출	`0`	`1`	`1`
두 번째 호출	`1`	`2`	`3`
세 번째 호출	`3`	`3`	`6`

네 번째 호출	`6`	`4`	`10`
다섯번째 호출	`10`	`5`	`15`

요약

지금까지 살펴본 배열 메서드를 요약해보도록 하자.

- 요소를 더하거나 지우기
  - `push(...items)` – 맨 끝에 요소 추가하기
  - `pop()` – 맨 끝 요소 추출하기
  - `shift()` – 첫 요소 추출하기
  - `unshift(...items)` – 맨 앞에 요소 추가하기
  - `splice(pos, deleteCount, ...items)` – `pos`부터 `deleteCount`개의 요소를 지우고, `items` 추가하기
  - `slice(start, end)` – `start`부터 `end` 바로 앞까지의 요소를 복사해 새로운 배열을 만듦
  - `concat(...items)` – 배열의 모든 요소를 복사하고 `items`를 추가해 새로운 배열을 만든 후 이를 반환함. `items`가 배열이면 이 배열의 인수를 기존 배열에 더해줌
- 원하는 요소 찾기
  - `indexOf/lastIndexOf(item, pos)` – `pos`부터 원하는 `item`을 찾음. 찾게 되면 해당 요소의 인덱스를, 아니면 `-1`을 반환함
  - `includes(value)` – 배열에 `value`가 있으면 `true`를, 그렇지 않으면 `false`를 반환함
  - `find/filter(func)` – `func`의 반환 값을 `true`로 만드는 첫 번째/전체 요소를 반환함
  - `findIndex`는 `find`와 유사함. 다만 요소 대신 인덱스를 반환함
- 배열 전체 순회하기
  - `forEach(func)` – 모든 요소에 `func`을 호출함. 결과는 반환되지 않음
- 배열 변형하기
  - `map(func)` – 모든 요소에 `func`을 호출하고, 반환된 결과를 가지고 새로운 배열을 만듦
  - `sort(func)` – 배열을 정렬하고 정렬된 배열을 반환함
  - `reverse()` – 배열을 뒤집어 반환함
  - `split/join` – 문자열을 배열로, 배열을 문자열로 변환함
  - `reduce(func, initial)` – 요소를 차례로 돌면서 `func`을 호출함. 반환값은 다음 함수 호출에 전달함. 최종적으로 하나의 값이 도출됨

분명 배웠던 내용이 너무 많아서 배열 관련 메소드를 전부 외울 수는 없을 것이다. 다만 위와같은 기능을 써야하는 경우가 올 때 해당 책을 다시 봐서 참고하도록 하자.

문제 1

"my-short-string"같이 여러 단어를 대시(-)로 구분한 문자열을 카멜 표기법을 사용한 문자열 "myShortString"로 변경해보자.

```
const word = 'backgroundColor'
```

힌트: `split`을 사용해 문자열을 배열로 바꾼 다음 `join`을 사용해 다시 합치면 됩니다.

문제 2

중요도: 4

배열 `arr`의 요소 중 `1`이상 `4` 이하 범위에 속하는 요소만 골라 새로운 배열에 집어넣고 해당 요소를 출력해 봅시다.

```
let arr = [5, 3, 8, 1];

/* 로직 */

let ret = /* 로직  */

alert( ret ); // 3,1 (조건에 맞는 요소)

alert( arr ); // 5,3,8,1 (기존 배열은 변경되지 않았습니다.)
```

문제 3

문제 2를 원본배열을 삭제하는 방식으로 해결해보세요

```
let arr = [5, 3, 8, 1];

/* 로직 */

let ret = /* 로직  */

alert( ret ); // 3,1 (조건에 맞는 요소)

alert( arr ); // 5,3,8,1 (기존 배열은 변경되지 않았습니다.)
```

문제 4

배열을 내림차순 해보세요

```
let arr = [5, 2, 1, -10, 8];

// 요소를 내림차순으로 정렬해주는 코드를 여기에 작성해보세요.

alert( arr ); // 8, 5, 2, 1, -10
```