자료구조와 자료형

배열

- **순서가 있는 컬렉션**이 필요할 때 배열을 사용
- 순서가 있는 컬렉션을 다뤄야 할 때 객체를 사용하면 **순서와 관련된 메서드가 없어 불편**
- **빈 배열을 선언**하는 두 가지 방법

```
// Array 생성자 함수 호출
let arr = new Array();
// 보통 사용하는 방법 (대괄호)
let arr = [];
```

- 초기에 포함할 요소들을 배열 선언하며 전달 가능

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

- 각 배열 요소엔 **0부터 시작하는 숫자(인덱스)**가 매겨져 있으며 이 숫자들은 **배열 내 순서**를 나타냄

```
let fruits = ["사과", "오렌지", "자두"];

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

- 인덱스를 통해 접근하여 요소 내용 수정 가능

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

- 인덱스를 통해 접근하여 새로운 요소를 배열에 추가하는 것도 가능

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

- length 속성을 통해 배열에 담긴 요소가 몇 개인지 알아낼 수 있음
- 배열 요소의 **자료형엔 제약이 없음** (타 언어처럼 정수만 저장할 수 있는 int 타입 배열 <= 이런 개념이 없음)

```
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.
```

- 배열의 마지막 요소는 쉼표로 끝날 수 있음 (실수 방지)

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

pop, push, shift, unshift

```
// pop : 배열의 맨 뒷 요소를 제거하며 반환
let fruits = ["사과", "오렌지", "배"];
alert( fruits.pop() ); // 배열에서 "배"를 제거하고 제거된 요소를 얼럿창에 띄웁니다.
alert( fruits ); // 사과,오렌지

// push : 배열의 맨 뒤에 요소를 추가
fruits = ["사과", "오렌지"];
fruits.push("배");
alert( fruits ); // 사과,오렌지,배

// shift : 배열의 맨 앞 요소를 제거하며 반환
fruits = ["사과", "오렌지", "배"];
alert( fruits.shift() ); // 배열에서 "사과"를 제거하고 제거된 요소를 얼럿창에 띄웁니다.
alert( fruits ); // 오렌지,배

// unshift : 배열의 맨 앞에 요소를 추가
fruits = ["오렌지", "배"];
fruits.unshift('사과');
alert( fruits ); // 사과,오렌지,배
```

- push, unshift는 **가변 인자**를 받는 메서드이므로 여러개의 값을 전달 가능

```
let arr = [];
arr.push(4, 5, 6);
console.log(arr); // [4, 5, 6]
arr.unshift(1, 2, 3);
console.log(arr); // [1, 2, 3, 4, 5, 6]
```

- 객체이므로 기존의 배열값을 변수에 대입할 경우 **참조 복사**가 진행

```
let fruits = ["바나나"];

// 참조를 복사함(두 변수가 같은 객체를 참조)
let arr = fruits;

// true
alert( arr === fruits );

// 참조를 이용해 배열을 수정합니다.
arr.push("배");

// 바나나,배 - 요소가 두 개가 되었습니다.
alert( fruits );
```

- 잘못된 배열 사용 방법

- 데이터를 추가할 경우 가급적 인덱스를 통해서 추가하지 말고 push, unshift 같은 메서드 쓰기

```
1. arr.test = 5 같이 숫자가 아닌 값을 프로퍼티 키로 사용하는 경우
2. arr[0]과 arr[1000]만 추가하고 그사이에 아무런 요소도 없는 경우
3. arr[1000], arr[999]같이 요소를 역순으로 채우는 경우
```

반복문

- 배열을 순회하기 위해서 for 반복문 사용 가능

```
let arr = ["사과", "오렌지", "배"];

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

- **for ... of 반복문**도 사용 가능

```
let fruits = ["사과", "오렌지", "자두"];

// 배열 요소를 대상으로 반복 작업을 수행합니다.
for (let fruit of fruits) {
    alert( fruit );
}
```

- 객체를 순회하는 것이 아니므로 for ... in 반복문을 이용하여 배열을 순회하는 것은 권장되지 않음
    - for .. in 반복문은 모든 "프로퍼티"를 대상으로 순회 (즉, 요소뿐만 아니라 프로퍼티도 조회됨)

```
let arr = [1, 2, 3];
arr.hello = "world";

for(let prop in arr) alert(prop); // 0, 1, 2, world 출력
```

length 프로퍼티

- length 속성값 => **배열의 가장 큰 인덱스에 1을 더한 값**
- length 속성에 값 대입 가능
- length 값을 감소시키면 감소시킨만큼 배열의 내용이 잘려나가게 됨

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

// 요소 2개만 남기고 자르기
arr.length = 2;
// [1, 2]
alert( arr );

// 본래 길이로 되돌리기
arr.length = 5;
// undefined: 삭제된 기존 요소들은 복구되지 않음
```

```
alert( arr[3] );

// 배열 비우기
arr.length = 0;
```

new Array()

- (new 키워드와 함께) Array를 생성자 함수로 호출하여 배열 생성 가능
    - 대괄호를 이용한 배열 리터럴에 비해서 잘 사용되지 않는 방법

```
let arr = new Array("사과", "배", "기타");
// 위의 코드와 같은 역할을 하는 배열 리터럴
// let arr = ["사과", "배", "기타"];

// 숫자 2가 포함된, 요소가 1개인 배열을 만드는게 아니고 "길이가 2인 비어있는 배열" 생성
arr = new Array(2);

// 모두 undefined 출력 (요소가 하나도 없는 배열)
alert( arr[0] );
alert( arr[1] );
// 길이는 2
alert( arr.length );
```

다차원 배열

- 배열 내부에 배열 포함 가능

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

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

// Q) 9를 출력하기 위한 코드 작성
```

배열 비교

- 두 배열의 **내용이 논리적으로 같은지 비교**하기 위해서 ==, === 연산자를 사용하면 안 됨
    - 배열은 객체이므로 **비교 연산자(==, ===)를 사용 시 두 배열의 참조를 비교**함

```
let arr1 = [1, 2, 3];
let arr2 = [1, 2, 3];

console.log( arr1 == arr2 ); // false
console.log( arr1 === arr2 ); // false
```

- 반복문을 이용해서 개별 항목들을 모두 비교하는 방식으로 내용 비교를 진행해야 함

```
function isEqualArray(arr1, arr2) {
    if(arr1 === arr2) return true;
    if(arr1.length !== arr2.length) return false;

    let eq = true;
    for (let i = 0; i < arr1.length; i++) {
        if(arr1[i] !== arr2[i]) {
            eq = false;
            break;
        }
    }
    return eq;
}

let arr = [1, 2, 3];
console.log( isEqualArray(arr, arr) );
console.log( isEqualArray([1, 2, 3], [1, 2, 3]) );
console.log( isEqualArray([1, 2, 3], [1, 2, 3, 4]) );
```

배열과 메서드

요소 추가·제거 메서드

- 배열에 요소를 추가하고 제거하는 메서드

```
arr.push(...items)      - 맨 끝에 요소 추가
arr.pop()               - 맨 끝 요소 제거
arr.shift()             - 맨 앞 요소 제거
arr.unshift(...items)   - 맨 앞에 요소 추가
```

splice 메소드

- splice 메소드를 사용하면 **배열 요소의 추가, 삭제, 교체 작업을 모두 할 수 있음**

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

- 첫 번째 매개변수는 조작을 가할 첫 번째 요소를 가리키는 인덱스(index)
- 두 번째 매개변수는 deleteCount로, 제거하고자 하는 요소의 개수
- elem1, ..., elemN은 배열에 추가할 요소 (가변 인수)

```
let arr = ["I", "study", "JavaScript"];
// 인덱스 1부터 요소 한 개를 제거
arr.splice(1, 1);
// ["I", "JavaScript"]
alert( arr );

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

- splice는 **삭제된 요소로 구성된 배열을 반환**함

```
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 **deleteCount를 0으로 설정**하면 요소를 제거하지 않으면서 새로운 요소를 추가 가능

```
let arr = ["I", "study", "JavaScript"];
// 인덱스 2부터 0개의 요소를 삭제하고, "complex"와 "language"를 추가합니다.
arr.splice(2, 0, "complex", "language");
// "I", "study", "complex", "language", "JavaScript"
alert( arr );
```

- slice 메서드 뿐만 아니라 배열 관련 메서드엔 **음수 인덱스 사용 가능**
  - 마이너스 부호 앞의 숫자는 **배열 끝에서부터 센 요소 위치**

```
let arr = [1, 2, 5];
// 인덱스 -1부터 (배열 끝에서부터 첫 번째 요소) 0개의 요소를 삭제하고 3과 4를 추가합니다.
arr.splice(-1, 0, 3, 4);
// 1,2,3,4,5
alert( arr );
```

slice 메소드

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

- slice 메서드는 **start 인덱스부터 (end를 제외한) end 인덱스까지의 요소를 복사**한 새로운 배열을 반환
  - start와 end는 둘 다 음수일 수 있는데 이땐, 배열 끝에서부터의 요소 개수를 의미

```
let arr = ["t", "e", "s", "t"];
// 인덱스가 1인 요소(e)부터 인덱스가 3인 요소(s)까지를 복사(인덱스가 3인 요소는 제외)
alert( arr.slice(1, 3) ); // e,s
// 인덱스가 -2인 요소(s)부터 제일 끝 요소까지를 복사
alert( arr.slice(-2) ); // s,t
// 인덱스가 -4인 요소(t)부터 인덱스가 -2인 요소(s)까지를 복사
alert( arr.slice(-4, -2) ); // t,e
```

- slice 메소드에 인수를 하나도 넘기지 않고 호출하여 **배열의 복사본**을 만들 수 있음

```
let arr = ["t", "e", "s", "t"];
let copied = arr.slice();

console.log(copied); // ["t", "e", "s", "t"]
console.log(arr === copied); // false
```

concat 메소드

- 기존 배열에 요소를 추가하여 **새로운 배열을 만들 때 사용** (**기존 배열에는 영향을 주지 않음**을 유의)
    - 인수엔 **배열이나 값**이 올 수 있으며 인수 개수엔 제한이 없음 (가변 인수)
        - 배열이 전달된 경우 전달된 **배열 내부의 모든 값들이 반환될 배열에 추가**됨 (**배열 자체가 추가되는 것이 아님**을 유의)

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

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

// arr의 요소 모두와 [3,4]의 요소 모두를 한데 모은 새로운 배열 반환
// 배열([3,4])이 추가되는 것이 아니고 값(3, 4)이 추가 됨을 유의
let c1 = arr.concat([3, 4])
alert( arr ); // 1,2 (기존 배열의 내용에는 영향 X)
alert( c1 ); // 1,2,3,4 (1,2,[3,4]가 아님!)

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

forEach로 반복작업 하기

- 주어진 함수를 **배열 요소 각각에 대해 실행**하기 위해서 사용하는 메서드
    - 이후에 배울 map 메서드가 배열 요소 각각의 **내용을 "변형"**하기 위해서 사용된다면, forEach는 **값을 "순회"**하기 위해서 사용한다는 의미가 더 강함

```
// forEach 메소드로 콜백 함수 전달
// 함수에서는 요소, 인덱스, 배열 정보에 접근 가능
arr.forEach(function(item, index, array) {
    // 요소에 무언가를 할 수 있습니다.
});

// 요소 모두를 얼럿창을 통해 출력해주는 코드
["Bilbo", "Gandalf", "Nazgul"].forEach(alert);

// 인덱스 정보까지 더해서 출력해주는 좀 더 정교한 코드
["Bilbo", "Gandalf", "Nazgul"].forEach((item, index, array) => {
    alert(`${item} is at index ${index} in ${array}`);
});
```

배열 탐색하기

배열 내에서 무언가를 찾고 싶을 때 쓰는 메서드

indexOf, lastIndexOf, includes

- arr.indexOf(item, from)는 인덱스 from부터 시작해 item(요소)을 찾으며, 요소를 발견하면 **해당 요소의 인덱스를 반환하고, 발견하지 못했으면 -1을 반환**
- arr.lastIndexOf(item, from)는 위 메서드와 동일한 기능을 하는데, **검색을 끝에서부터 시작**함
- arr.includes(item, from)는 인덱스 from부터 시작해 item이 있는지를 검색하는데, 해당하는 **요소를 발견하면 true를 반환**
    - 요소의 위치를 정확히 알고 싶은게 아니고 **요소가 배열 내 존재하는지 여부만 확인**하고 싶다면 arr.includes를 사용하는 것이 권장됨
    - https://stackoverflow.com/questions/35370222/array-prototype-includes-vs-array-prototype-indexof
- indexOf, lastIndexOf 메서드는 요소를 찾을 때 내부적으로 **완전 항등 연산자(===)을 사용**함

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

alert( arr.indexOf(0) ); // 1
alert( arr.indexOf(false) ); // 2 (false를 0과 비교하거나 하지 않음)
alert( arr.indexOf(null) ); // -1

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

- 단, includes는 SameValueZero 알고리즘을 적용하여 **NaN 값을 찾는 작업도 제대로 처리**함

```
const arr = [NaN];
// 완전 항등 비교 === 는 NaN엔 동작하지 않으므로 0이 출력되지 않음
alert( arr.indexOf(NaN) ); // -1
// includes 메서드를 이용하면, NaN의 존재 여부를 확인 가능함
alert( arr.includes(NaN) ); // true
```

find, findIndex

- find 메소드로 **검색 로직이 담긴 함수를 전달**
  - 함수 내부에서는 **검색 조건이 맞는 경우 true를 반환**하도록 코드 작성
  - 요소를 찾지 못한 경우에는 undefined를 반환 (직접 코드로 undefined를 반환하는게 아니고 true를 반환하지 않은 상태로 탐색이 끝나면 자동으로 undefined 반환)

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

```
let users = [
    {id: 1, name: "John"},
    {id: 2, name: "Pete"},
    {id: 3, name: "Mary"},
    {id: 1, name: "Smith"},
];

// 맨 첫 번째로 조건을 만족하는 John만 반환 (Smith는 반환하지 않음을 유의!)
let user = users.find(item => item.id === 1);

alert(user.name); // John
```

- findIndex는 find와 동일한 작업을 하지만, 조건에 맞는 요소를 반환하는 대신 **해당 요소의 인덱스를 반환**

```
let idx = users.findIndex(item => item.name === "Smith");
alert(idx); // 인덱스(3)를 반환
```

filter

- **조건을 충족하는 요소 여러 개를 반환**받기 위해서는 filter 메소드 사용
  - filter는 find와 문법이 유사하지만, **조건에 맞는 요소 전체를 담은 배열을 반환**

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

```
let users = [
    {id: 1, name: "John"},
    {id: 2, name: "Pete"},
    {id: 3, name: "Mary"}
];

// id가 3보다 작은 모든 사용자 반환 (앞쪽 사용자 두 명을 반환)
let someUsers = users.filter(item => item.id < 3);

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

배열을 변형하는 메서드

map

- **배열 요소 전체를 대상으로 함수를 호출**하고, 함수 호출 결과를 배열로 반환

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

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

let users = [
    {id: 1, name: "John"},
    {id: 2, name: "Pete"},
    {id: 3, name: "Mary"}
];
let altered = users.map(user => {
    return {
        id: user.id * 100,
        name: user.name.toUpperCase(),
        idName: `${user.id}_${user.name}`
    };
});
console.log(altered);
```

sort

- 배열의 요소를 정렬하는 메소드
    - **기존 배열의 내용이 수정**됨을 유의
    - 기본적으로 요소는 **문자열로 취급되어 재정렬**됨을 유의
        - **비교 함수를 직접 전달**하여 비교 방법을 바꿀 수 있음
        - 일반적으로 퀵소트 알고리즘을 이용하여 정렬 작업을 진행하지만, 개발자는 따로 내부 정렬 동작 원리를 알 필요 없이 **비교함수만 잘 작성해서 전달**하면 됨

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

// arr 재정렬
arr.sort();

// 1, 15, 2
// 기본적으로 모든 요소는 "문자형으로 변환된 이후에 재정렬"되므로 사전편집 순으로 비교가 진행되고, 2는 15보다 큰 값으로 취급됨 (숫자가 정렬된다고 예상했다면 이상한 결과)
alert( arr );
```

- sort 메소드로 전달할 **정렬 함수(정렬 기준을 정의해주는 함수, ordering function) 직접 정의 가능**

```
function compare(a, b) {
    if (a > b) return 1; // 첫 번째 값이 두 번째 값보다 큰 경우 "양수"를 반환
    if (a == b) return 0; // 두 값이 같은 경우 0을 반환
    if (a < b) return -1; // 첫 번째 값이 두 번째 값보다 작은 경우 "음수"를 반환
}
```

```
function compareNumeric(a, b) {
    if (a > b) return 1;
    if (a == b) return 0;
    if (a < b) return -1;
}

let arr = [ 1, 2, 15 ];
arr.sort(compareNumeric);
// 1, 2, 15
alert(arr);
```

- 양수와 음수를 반환할 수 있다는 성질을 이용하면 더 축약하여 함수 작성 가능

```
let arr = [ 1, 2, 15 ];
// 정렬 함수는 어떤 숫자든 반환 가능
// (양수를 반환하는 경우 첫 번째 인수가 두 번째 인수보다 '크다'를 나타내고, 음수를 반환하는 경우 첫 번째 인수가 두 번째 인수보다 '작다'를 나타냄)
arr.sort(function(a, b) { return a - b; });
// 1, 2, 15
alert(arr);
```

- 화살표 함수를 사용하여 정렬 함수 작성 가능 (더 간결하게 작성 가능)

```
arr.sort( (a, b) => a - b );
```

실습 문제

```
// Q1) 영화 제목 길이 오름차순으로 정렬
let movies = ["누구를 위하여 종은 울리나", "기생충", "패왕별희", "북북서로 진로를 돌려라", "시"];
// Q2) 학생 점수 내림차순으로 정렬
let students = [{ name: "김철수", score: 100 }, { name: "박미림", score: 80 }, { name: "유영희", score: 90 }];
```

실습 풀이

```
movies.sort((m1, m2) => m1.length - m2.length);
students.sort((s1, s2) => s2.score - s1.score);

console.log(movies);
console.log(students);
```

reverse

- reverse는 배열의 요소를 **역순으로 정렬**

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

split, join

- split 메서드는 **구분자(delimiter)를 기준으로 문자열을 쪼갠 배열**을 반환

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

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

// ["Bilbo","Gandalf","Nazgul"]
console.log(arr);

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

- split 메서드의 인수로 빈 문자열을 전달한 경우 문자열을 **글자 단위로 분리** 가능

```
let str = "test";

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

- join 메서드는 split과 반대 역할을 하는 메서드로 **배열 요소를 모두 합친 후 하나의 문자열을 생성**

```
let arr = ['Bilbo', 'Gandalf', 'Nazgul'];
// 배열 요소 모두를 ;를 사용해 하나의 문자열로 합칩니다.
let str = arr.join(';');
// "Bilbo;Gandalf;Nazgul"
alert( str );
// "Bilbo, Gandalf, Nazgul"
alert( arr.join(', ') );
```

Array.isArray로 배열 여부 알아내기

- 자바스크립트에서 **배열은 독립된 자료형으로 취급되지 않고 객체로 취급**됨. 따라서, typeof 연산자로는 일반 객체와 배열을 구분할 수 없음
- 배열은 자주 사용되는 자료구조이기 때문에 **배열인지 아닌지를 감별해내는 특별한 메서드 Array.isArray**가 존재함

```
alert(Array.isArray({})); // false
alert(Array.isArray([])); // true
```