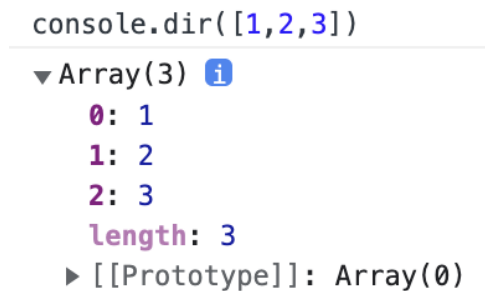


# 배열

🕒 작성일시	@June 1, 2023 9:29 PM
📎 자료	
# 주차	2

## 자바스크립트 배열 정의

```
console.dir([1,2,3])
```



- 배열?  
정수 **인덱스**(오프셋)을 이용해 각 요소에 접근할 수 있게 **순차적으로 요소를 저장한 집합체**
- 자바스크립트의 배열은 객체지만 내부적으로 특화된 객체이므로 배열로 취급
- 정수 인덱스를 키(객체의 프로퍼티 이름, 식별자)로 객체 데이터 오프셋 표현(정수키값은 객체에서 문자열로 변환됨)
- Array 프로토타입을 상속받아 자바스크립트에서 제공하는 Array 프로퍼티, 메서드를 이용가능

## 배열 사용하기

### 배열 생성

#### 1. 배열리터럴 \*

```
const numbers = [];
```

```
// 내장된 length 프로퍼티 이용해 배열의 길이 확인
console.log(numbers.length); // 0
```

#### 2. Array 생성자 함수

```
const num1 = new Array();
console.log(num1.length); // 0

// 1개 인자 제공 -> Array 생성자에 배열 길이 지정
const num2 = new Array(2);
console.log(num2, num2.length); // [비어 있음 × 2] 2

// Array 생성자의 인자에 요소 집합을 제공
const num3 = new Array(1, 2, 3, 4, 5);
console.log(num3); // (5) [1, 2, 3, 4, 5]
```

다른 언어와 달리 자바스크립트에서는 한 배열이 다양한 종류의 요소를 포함할 수 있다.(대부분의 스크립트 언어가 마찬가지다.)

```
const objs = [1, "str", true, null];
```

**Array.isArray()** 메서드를 이용해 특정 객체가 배열인지 여부 확인 가능

```
const num = 3;
const arr = [1, 2];

console.log(Array.isArray(num)); // false
console.log(Array.isArray(arr)); // true
```

## 배열 요소 접근 및 값 수정

`arr[index]` 로 값에 접근 가능

```
// 배열 요소 값 할당
const nums = [];
for(let i=0; i<4; i++) {
  nums[i] = i+1;
}
```

```
// 배열의 요소의 누적합
const nums = [1, 2, 3, 4, 5];
let sum = 0;
for(let i=0; i<nums.length; i++) {
  sum += nums[i];
}
console.log(sum);

// nums.reduce((acc, cur) => acc+cur, 0);
```

## 문자열로 배열 만들기

`String.prototype.split(separator)` : 지정한 구분자를 기준으로 문자열을 분리하여 배열을 생성

```
// 쉼표의 개수 구하기
const str = "a,b,c";
const countComma = str.split(',').length - 1; // 2
```

## 얕은 복사 vs 깊은 복사

```
const nums = [1,2,3];
const copy = nums;
console.log(copy); // [1,2,3]
nums[0] = 0;
console.log(copy); // [0,2,3]
```

위의 예제처럼 배열을 다른 배열로 할당할 때 실제로는 원본 배열의 레퍼런스(참조값) 할당하는 것. 따라서 원본 배열을 바꾸면 할당된 배열도 바뀐다.

원시값 → immutable → 값에 의한 전달

객체 → mutable → 참조에 의한 전달

이와 같은 동작은 **얕은 복사(shallow copy)**라고 한다.

원본 배열 요소를 새로운 배열 요소로 복사하는 **깊은 복사(deep copy)**가 필요할 때가 있다.

## 깊은 복사 방법

```
const original = [1,2,3];
const copy = [];

function deepCopy(original, copy) {
  for(let i=0; i<original.length; i++) {
    copy[i] = original[i];
  }
}

deepCopy(original, copy);
console.log(copy); // [1,2,3]
original[0] = 0;
console.log(copy); // [1,2,3]
```

## 깊은 복사 방법 (1차원 배열!)

```
const original = [1,2,3];  
// spread operator  
const copy = [...original];
```

```
const original = [1,2,3];  
// slice()  
const copy = original.slice();
```

## 접근자 함수

자바스크립트는 배열 요소에 접근할 수 있는 다양한 함수를 제공한다. 이들을 접근자 함수(accessor function)라 부른다

## 값 검색 indexOf, lastIndexOf

```
const str = ['a', 'b', 'c'];  
console.log(str.indexOf('a')); // 0  
console.log(str.indexOf('d')); // -1
```

**Array.indexOf(searchEl):** 인자로 제공한 값이 배열에 존재하는 지 체크, 인자 값이 배열에 있으면 해당 인덱스 위치 반환, 값이 배열에 없으면 -1 반환

**Array.lastIndexOf(searchEl):** 주어진 값과 일치하는 부분을 **fromIndex**로부터 역순으로 탐색하여, 최초로 마주치는 인덱스를 반환합니다. 일치하는 부분을 찾을 수 없으면 **-1**을 반환

## 배열을 문자열로 표현하기

```
const names = ['john', 'james', 'jack'];  
// 배열의 요소를 콤마로 구분하는 문자열 반환  
console.log(names.join()); // john,james,jack  
console.log(names.toString()); // john,james,jack
```

## 기존 배열 이용해 새배열 만들기

```
// 기존배열.concat(합칠배열)
const arr1 = [1, 2, 3];
const arr2 = [4, 5];
const concatArr = arr1.concat(arr2);
console.log(concatArr); //(5) [1, 2, 3, 4, 5]
```

```
// splice(): 배열의 기존 요소를 삭제 또는 교체 하거나 새 요소를 추가하여 배열의 내용을 변경
const arr = [1,2,3];
// 기존 요소 삭제
arr.splice(0,1); //[1]
console.log(arr); //[2,3]

// 교체
arr.splice(0,1,5);
console.log(arr); //[5,3]

// 추가
arr.splice(0,0,9);
console.log(arr); //[9,5,3]
```

## 변형자 함수

### 배열에 요소 추가

```
// 배열 끝에 요소 추가하기
// push()
const arr = [0,2];
arr.push(3);
console.log(arr); //[0,2,3];
```

```
// 배열 앞에 요소 추가하기
// unshift()
const arr = [1,2];
arr.unshift(0);
console.log(arr); //[0,1,2]
```

### 배열의 요소 삭제

```
// 배열의 마지막 요소 제거
// pop()
const arr = [1,2];
arr.pop();
console.log(arr); //[1]
```

```
// 배열의 맨 처음 요소 제거
// shift()
const nums = [6,1,2,3,4,5];
const first = nums.shift();
nums.push(first);
console.log(nums); // [1,2,3,4,5,6]
```

## 배열 중간에 요소를 추가하거나 배열의 중간에 있는 요소 삭제하기

`splice(startIndex, deleteCount, ..items)`

- `startIndex`: 배열에 요소 추가/삭제/교체할 시작지점
- `deleteCount`: 삭제할 요소의 개수(추가시 0)
- `items`: 배열에 추가할 요소들

```
// add
const nums = [1,2,3,7,8,9];
const newArr = [4,5,6];
nums.splice(3,0,...newArr);
console.log(nums); // (9) [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
// delete
const nums = [1,2,3,100,200,300,400,4,5];
nums.splice(3,4);
console.log(nums); // (5) [1, 2, 3, 4, 5]
```

## 배열 요소 정렬

```
// Array.reverse(): 배열의 요소를 역순으로 바꿈
const nums = [1,2,3];
nums.reverse();
console.log(nums); // [3,2,1]
```

### `sort([compareFunction])`

배열의 요소를 정렬한 후 반환. 원본 배열도 변경됨

`compareFunction(a,b)`

- `result < 0` : a가 먼저옴, a를 b보다 낮은 인덱스로 소트
- `result == 0` : a,b 숫자가 같음
- `result > 0` : a가 b보다 큼, b를 a보다 낮은 인덱스로 소트

```
// 오름차순 정렬
[3,2,1].sort((a,b) => {
  console.log('a', a, 'b', b, 'a-b', a-b)
  return a-b
});
// a 2 b 3 a-b -1
// a 1 b 2 a-b -1
```

```
const months = ['March', 'Jan', 'Feb', 'Dec'];
// 기본 정렬 순서는 문자열 유니코드 따름
months.sort();
console.log(months);
// Expected output: Array ["Dec", "Feb", "Jan", "March"]
```

## 반복자 함수

```
arr.forEach(callback(currentvalue[, index[, array]]), thisArg)
```

각 배열 요소에 대해 한 번씩 callback 함수 실행

```
function square(num) {
  console.log(num*num);
}
const nums = [1,2,3];
nums.forEach(square)
```

```
// every(): 배열 안의 모든 요소가 주어진 판별함수를 통과하면 true반환, 아니면 false 반환
function isEven(num) {
  return num%2 === 0;
}

const nums = [1,2,3,4];
console.log(nums.every(isEven)); // false
```

```
// some(): 배열 안의 요소 중 하나 요소라도 주어진 판별함수 만족하면 true 반환
function isEven(num) {
  return num%2 === 0;
}

const nums = [1,2,3,4];
console.log(nums.some(isEven)); // true
```

`reduce()` 메서드는 배열의 각 요소에 대해 주어진 리듀서 (reducer) 함수를 실행하고, 하나의 결과값을 반환

```
arr.reduce(callback[, initialValue])
```

리듀서 함수는 네 개의 인자를 가짐

1. acc(누산기)
2. cur(현재 값)
3. idx(현재인덱스)
4. 원본배열

리듀서 함수의 반환값은 acc에 할당되고, acc는 순회중 유지 되므로 최종 결과는 하나의 값이 됨.

```
[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

`reduceRight()` 메서드는 누적기에 대해 함수를 적용하고 배열의 각 값 (오른쪽에서 왼쪽으로)은 값을 단일 값으로 줄여야합니다.

## 새 배열을 반환하는 반복자 함수

`map()` 메서드는 배열 내의 모든 요소 각각에 대하여 주어진 함수를 호출한 결과를 모아 새로운 배열을 반환합니다.

```
arr.map(callback(currentValue[, index[, array]]), thisArg)
```

`map` 은 호출한 배열의 값을 변형하지 않습니다. 단, `callback` 함수에 의해서 변형될 수는 있습니다.

```
const nums = [1,2,3];  
const newNums = nums.map(item => item*2);  
console.log(newNums); // (3) [2, 4, 6]
```



**filter()** 메서드는 주어진 함수의 테스트를 통과하는 모든 요소를 모아 새로운 배열로 반환. 호출되는 배열 변화시키지 않음

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

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

## 이차원 배열과 다차원 배열

자바스크립트는 기본적으로 일차원 배열만 지원한다. 하지만 배열의 배열을 이용해 다차원 배열을 만들수 있다.

### 이차원 배열 만들기

이차원 배열은 행과 열을 가진 스프레드시트 같은 구조다.

자바스크립트에서는 `const arr = []` 혹은 `const arr[][]`와 같은 방법으로 선언할 수 없습니다.

```
const matrix = [
  [1,2,3],
  [1,2,3],
  [1,2,3]
];
```

```
// matrix[n] 행
// matrix[n][m] 열
```

```
// 1. 배열에 초기값을 할당하는 방법
const arr = [[1,2], [3,4]];

// 2. 배열을 인자로 전달하는 방법
const arr = [];
arr.push([1,2]); // [[1,2]]
arr.push([2,3]); // [[1,2], [2,3]]

// 3. 반복문으로 배열안에 배열 생성
const row = 2;
const col = 2;

const arr = new Array(row);
```

```

console.log(arr); // (2) [empty × 2]
for(let i=0;i<row; i++) {
    arr[i] = new Array(col);
}

console.log(arr); // (2) [Array(2), Array(2)]

// 4. ES6문법
// empty 를 가진 빈 배열의 경우 map함수가 제대로 실행되지 않음 -> fill으로 값 채워줌
const arr = new Array(row).fill(0).map(() => new Array(col));

```

## 이차원 배열 요소 처리하기

이차원 배열 요소는 두 가지 주요 패턴으로 처리한다.

1. 배열의 열을 기준
2. 배열의 행을 기준

```

// 열 기준 처리
// 외부 루프가 행을 처리하고 안쪽 루프가 열을 처리
// 각 행은 한 학생의 점수 집합을 포함
const grades = [[20,30], [80,90]];
let total = 0;
let average = 0;

for(let row=0; row<grades.length; row++) {
    for(let col=0; col<grades[row].length; col++) {
        total += grades[row][col];
    }
    average = total/grades[row].length;
    console.log(`학생 ${row+1}의 성적평균은 ${average.toFixed(2)}`);
    total = 0;
    average = 0
}

// 학생 1의 성적평균은 25.00
// 학생 2의 성적평균은 85.00

```

```

// 행 기준 처리
// 외부 루프가 열을 처리하고 안쪽 루프가 행을 처리
// 각 열은 과목의 점수
const grades = [[20,30], [80,90]];
let total = 0;
let average = 0;

for(let col=0; col<grades.length; col++) {
    for(let row=0; row<grades[col].length; row++) {
        total += grades[row][col];
    }
    average = total/grades[col].length;
    console.log(`과목 ${col+1}의 성적평균은 ${average.toFixed(2)}`);
    total = 0;
    average = 0
}

```

```
// 과목 1의 성적평균은 50.00  
// 과목 2의 성적평균은 60.00
```

## 들쭉날쭉한 배열

배열의 행이 포함하는 요소의 개수가 서로 다른 배열.

하지만 자바스크립트는 모든 행의 길이를 정확하게 알 수 있으므로 이러한 배열도 쉽게 처리

→ `arr[row].length`

```
const grades = [[20,30], [80,90,100]];
let total = 0;
let average = 0;

for(let row=0; row<grades.length; row++) {
  for(let col=0; col<grades[row].length; col++) {
    total += grades[row][col];
  }
  average = total/grades[row].length;
  console.log(`학생 ${row+1}의 성적평균은 ${average.toFixed(2)}`);
  total = 0;
  average = 0
}
```

## 객체를 요소로 포함하는 배열

배열은 객체 요소도 포함할 수 있다.

배열 안의 객체 값 가져오기 → `arr[index].property`

```
const pens = [
  {color: 'red', price: 5000, brand: 'monami'},
  {color: 'blue', price: 3000, brand: 'paker'},
  {color: 'green', price: 1000}
];
// 배열의 요소인 객체의 프로퍼티 기준 오름차순 정렬
pens.sort((a,b) => a.price-b.price);
// pens
// 0: {color: 'green', price: 1000}
// 1: {color: 'blue', price: 3000, brand: 'paker'}
// 2: {color: 'red', price: 5000, brand: 'monami'}
```

## 객체에 포함된 배열

객체에 복잡한 데이터를 저장할 때 배열을 활용할 수 있다.

```

function weekTemps () {
  this.dataStore = [];
  this.add = function(temp) {
    this.dataStore.push(temp);
  };
  this.average = function() {
    let total = 0;
    for(let i=0; i<this.dataStore.length; i++) {
      total += this.dataStore[i];
    }
    return total/this.dataStore.length;
  }
}
const thisWeekTemp = new weekTemps();

thisWeekTemp.add(52);
thisWeekTemp.add(60);
thisWeekTemp.add(58);
thisWeekTemp.add(49);
thisWeekTemp.add(50);
thisWeekTemp.add(52);
thisWeekTemp.add(53);

console.log(thisWeekTemp.dataStore); // [52, 60, 58, 49, 50, 52, 53]
console.log(thisWeekTemp.average());

```

## 연습 문제

```

// 1
function Grades() {
  this.grades = [];
  this.add = function(grade) {
    this.grades.push(grade);
  };
  this.average = function() {
    let total = this.grades.reduce((acc, cur) => acc+cur, 0);
    return total / this.grades.length;
  }
}

const obj = new Grades();
obj.add(20);
obj.add(80);
obj.average(); // 50

```

```

// 2
function printArr(arr) {
  console.log(arr.join());
}

function printReverseArr(arr) {
  console.log(arr.reverse().join());
}
printArr(['hi', 'js']); // hi,js
printReverseArr(['hi', 'js']); // js,hi

```

```
// 3
function weekTemps () {
  this.dataStore = new Array(4).fill(0).map(() => new Array(7));
  this.add = function(week, day, temp) {
    this.dataStore[week][day] = temp;
  };
  this.weekAverage = function(week) {
    let total = 0;
    for(let i=0; i<this.dataStore[week].length; i++) {
      total += this.dataStore[week][i];
    }
    return total/this.dataStore[week].length;
  },
}
const thisWeekTemp = new weekTemps();
```