

# ECMAScript 6 배열과 객체

# 배열 (Arrays)

## 배열의 정의와 특징

```
const numbers = [1, 2, 3, 4, 5]

console.log(numbers[0])    // 1
console.log(numbers[-1])   // undefined
console.log(numbers.length) // 5
```

```
const numbers = [1, 2, 3, 4, 5]

console.log(numbers[numbers.length - 1]) // 5
console.log(numbers[numbers.length - 2]) // 4
console.log(numbers[numbers.length - 3]) // 3
console.log(numbers[numbers.length - 4]) // 2
console.log(numbers[numbers.length - 5]) // 1
```

- 키와 속성들을 담고 있는 참조 타입의 객체(object)
- 순서를 보장하는 특징이 있음
- 주로 대괄호를 이용하여 생성하고, 0을 포함한 양의 정수 인덱스로 특정 값에 접근 가능
- 배열의 길이는 `array.length` 형태로 접근 가능
  - (참고) 배열의 마지막 원소는 `array.length - 1`로 접근

## 배열 관련 주요 메서드 목록 (1) – 기본편 (기본 배열 조작)

- (참고) 추가적인 배열 관련 메서드 정보는 아래 링크에서 참고
  - [MDN](#), [ECMA262](#)(#sec-properties-of-the-array-constructor)

메서드	설명	비고
reverse	원본 배열의 요소들의 순서를 반대로 정렬	
push & pop	배열의 가장 뒤에 요소를 추가 또는 제거	
unshift & shift	배열의 가장 앞에 요소를 추가 또는 제거	
includes	배열에 특정 값이 존재하는지 판별 후 참/거짓 반환	
indexOf	배열에 특정 값이 존재하는지 판별 후 인덱스 반환	요소가 없을 경우 -1 반환
join	배열의 모든 요소를 구분자를 이용하여 연결	구분자 생략 시 쉼표 기준

## 배열 관련 주요 메서드 - reverse

```
const numbers = [1, 2, 3, 4, 5]
numbers.reverse()
console.log(numbers) // [5, 4, 3, 2, 1]
```

### □ array.reverse()

- 원본 배열의 요소들의 순서를 반대로 정렬

## 배열 관련 주요 메서드 – push & pop

```
const numbers = [1, 2, 3, 4, 5]

numbers.push(100)
console.log(numbers) // [1, 2, 3, 4, 5, 100]

numbers.pop()
console.log(numbers) // [1, 2, 3, 4, 5]
```

### ❑ array.push()

- 배열의 가장 뒤에 요소 추가

### ❑ array.pop()

- 배열의 마지막 요소 제거

## 배열 관련 주요 메서드 – unshift & shift

```
const numbers = [1, 2, 3, 4, 5]

numbers.unshift(100)
console.log(numbers) // [100, 1, 2, 3, 4, 5]

numbers.shift()
console.log(numbers) // [1, 2, 3, 4, 5]
```

- `array.unshift()`
  - 배열의 **가장 앞에** 요소 추가
- `array.shift()`
  - 배열의 **첫번째** 요소 제거

## 배열 관련 주요 메서드 – includes

```
const numbers = [1, 2, 3, 4, 5]
console.log(numbers.includes(1)) // true
console.log(numbers.includes(100)) // false
```

### □ array.includes(value)

- 배열에 특정 값이 존재하는지 판별 후 참 또는 거짓 반환



## 배열 관련 주요 메서드 – indexOf

```
const numbers = [1, 2, 3, 4, 5]
let result

result = numbers.indexOf(3) // 2
console.log(result)

result = numbers.indexOf(100) // -1
console.log(result)
```

### □ array.indexOf(value)

- 배열에 특정 값이 존재하는지 확인 후 가장 첫번째로 찾은 요소의 인덱스 반환
- 만약 해당 값이 없을 경우 -1 반환

## 배열 관련 주요 메서드 – join

```
const numbers = [1, 2, 3, 4, 5]
let result

result = numbers.join()    // 1,2,3,4,5
console.log(result)

result = numbers.join('')  // 12345
console.log(result)

result = numbers.join(' ') // 1 2 3 4 5
console.log(result)

result = numbers.join('-') // 1-2-3-4-5
console.log(result)
```

### □ array.join([separator])

- 배열의 모든 요소를 연결하여 반환
- separator(구분자)는 선택적으로 지정 가능하며, 생략 시 쉼표를 기본 값으로 사용

## | 배열 실습 (1)

- 목표: 자바스크립트 배열 기본 조작 연습 (06-arrays.js)
- 문제: 파일에 작성된 주석 참고

## 배열 관련 주요 메서드 목록 (2) – 심화편 (Array Helper Methods)

- 배열을 순회하며 특정 로직을 수행하는 메서드
- 메서드 호출 시 인자로 callback 함수\*를 받는 것이 특징
  - callback 함수\*: 어떤 함수의 내부에서 실행될 목적으로 인자로 넘겨받는 함수를 말함

메서드	설명	비고
forEach	배열의 각 요소에 대해 콜백 함수를 한 번씩 실행	반환 값 없음
map	콜백 함수의 반환 값을 요소로 하는 새로운 배열 반환	
filter	콜백 함수의 반환 값이 참인 요소들만 모아서 새로운 배열을 반환	
reduce	콜백 함수의 반환 값들을 하나의 값(acc)에 누적 후 반환	
find	콜백 함수의 반환 값이 참이면 해당 요소를 반환	
some	배열의 요소 중 하나라도 판별 함수를 통과하면 참을 반환	
every	배열의 모든 요소가 판별 함수를 통과하면 참을 반환	

## (참고) Django로 보는 callback 함수 예시

```
# urls.py

from django.urls import path
from . import views


urlpatterns = [
    path('index/', views.index, name='index'),
]
```

```
# views.py

from django.shortcuts import render

def index(request):
    # ... 생략 ...
    return render(request, 'articles/index.html', context)
```

path 함수에 전달되는 callback 함수



## 배열 관련 주요 메서드 – forEach

```
array.forEach((element, index, array) => {  
  // do something  
})
```

```
const ssafy = ['광주', '대전', '구미', '서울']  
  
ssafy.forEach((region, index) => {  
  console.log(region, index)  
  // 광주 0  
  // 대전 1  
  // 구미 2  
  // 서울 3  
})
```

- array.forEach(callback(element[, index[, array]]))
  - 배열의 각 요소에 대해 콜백 함수를 한 번씩 실행
  - 콜백 함수는 3가지 매개변수로 구성
    - element: 배열의 요소
    - index: 배열 요소의 인덱스
    - array: 배열 자체
  - 반환 값(return)이 없는 메서드

## 배열 관련 주요 메서드 – map

```
array.map((element, index, array) => {  
  // do something  
})
```

```
const numbers = [1, 2, 3, 4, 5]  
  
const doubleNums = numbers.map((num) => {  
  return num * 2  
})  
  
console.log(doubleNums) // [2, 4, 6, 8, 10]
```

- array.map(callback(element[, index[, array]]))
  - 배열의 각 요소에 대해 콜백 함수를 한 번씩 실행
  - 콜백 함수의 반환 값을 요소로 하는 새로운 배열 반환
  - 기존 배열 전체를 다른 형태로 바꿀 때 유용

## 배열 관련 주요 메서드 – filter

```
array.filter((element, index, array) => {  
  // do something  
})
```

```
const numbers = [1,2,3]  
  
const oddNums = numbers.filter((num) => {  
  return num % 2  
})  
console.log(oddNums) // [1, 3, 5]
```

- array.filter(callback(element[, index[, array]]))
  - 배열의 각 요소에 대해 콜백 함수를 한 번씩 실행
  - 콜백 함수의 반환 값이 참인 요소들만 모아서 새로운 배열을 반환
  - 기존 배열의 요소들을 필터링할 때 유용



## 배열 관련 주요 메서드 – reduce

□ `array.reduce(callback(acc, element, [index[, array]]), initialValue)`

```
array.reduce((acc, element, index, array) => {  
  // do something  
}, initialValue)
```

- 배열의 각 요소에 대해 콜백 함수를 한 번씩 실행
- 콜백 함수의 반환 값들을 하나의 값(acc)에 누적 후 반환
- reduce 메서드의 주요 매개변수
  - `acc`: 이전 callback 함수의 반환 값이 누적되는 변수
  - `initialValue`: 최초 callback 함수 호출 시 `acc`에 할당되는 값으로, 선택적으로 설정 가능하며 직접 제공하지 않으면 배열의 첫번째 값 사용
- (참고) 빈 배열의 경우 `initialValue`를 제공하지 않으면 에러 발생

## 배열 관련 주요 메서드 – reduce 예시

□ `array.reduce(callback(acc, element, [index[, array]]), initialValue)`

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

const result = numbers.reduce((acc, num) => {
  return acc + num
}, 0)

console.log(result) // 6
```

## (참고) 배열 관련 주요 메서드 – reduce 동작 방식

□ `array.reduce(callback(acc, element, [index[, array]]), initialValue)`

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

const result = numbers.reduce((acc, num) => {
  return acc + num
}, 0)

console.log(result) // 6
```

↓

```
const numbers = [1, 2, 3]
const result = numbers.reduce((acc, num) => {
  return acc + num
}, 0)
```

acc: 0, num: 1  
0 + 1

↓

```
const numbers = [1, 2, 3]
const result = numbers.reduce((acc, num) => {
  return acc + num
}, 0)
```

acc: 1, num: 2  
1 + 2

↓

```
const numbers = [1, 2, 3]
const result = numbers.reduce((acc, num) => {
  return acc + num
}, 0)
```

acc: 3, num: 3  
3 + 3

## 배열 관련 주요 메서드 – find

```
arrays.find((element, index, array) => {  
  // do something  
})
```

```
const avengers = [  
  { name: 'Tony Stark', age: 45 },  
  { name: 'Steve Rogers', age: 32 },  
  { name: 'Thor', age: 40 },  
]  
  
const result = avengers.find((avenger) => {  
  return avenger.name === 'Tony Stark'  
})  
  
console.log(result) // {name: "Tony Stark", age: 45}
```

- array.find(callback(element[, index[, array]]))
  - 배열의 각 요소에 대해 콜백 함수를 한 번씩 실행
  - 콜백 함수의 반환 값이 참이면 해당 요소를 반환
  - 찾는 값이 배열에 없으면 undefined 반환

## 배열 관련 주요 메서드 – some

```
array.some((element, index, array) => {  
  // do something  
})
```

```
const numbers = [1, 3, 5, 7, 9]  
  
const hasEvenNumber = numbers.some((num) => {  
  return num % 2 === 0  
})  
console.log(hasEvenNumber) // false  
  
const hasOddNumber = numbers.some((num) => {  
  return num % 2  
})  
console.log(hasOddNumber) // true
```

- array.some(callback(element[, index[, array]]))
  - 배열의 요소 중 하나라도 주어진 판별 함수를 통과하면 참을 반환
  - 모든 요소가 통과하지 못하면 거짓 반환
  - (참고) 빈 배열은 항상 거짓 반환

## 배열 관련 주요 메서드 – every

```
array.every((element, index, array) => {  
  // do something  
})
```

```
const numbers = [2, 4, 6, 8, 10]  
  
const isEveryNumberEven = numbers.every((num) => {  
  return num % 2 === 0  
})  
console.log(isEveryNumberEven) // true  
  
const isEveryNumberOdd = numbers.every((num) => {  
  return num % 2  
})  
console.log(isEveryNumberOdd) // false
```

- array.every(callback(element[, index[, array]]))
  - 배열의 모든 요소가 주어진 판별 함수를 통과하면 참을 반환
  - 모든 요소가 통과하지 못하면 거짓 반환
  - (참고) 빈 배열은 항상 참 반환

## | 배열 실습 (2)

- 목표: 자바스크립트 배열 관련 심화 메서드 연습 (07-arrays-advanced.js)
- 문제: 파일에 작성된 주석 참고

## (참고) 배열 순회 방법 비교

```
const ssafy = ['광주', '대전', '구미', '서울']

// for loop
for (let i = 0; i <= ssafy.length; i++) {
  console.log(i, ssafy[i])
  // 0, 광주
  // 1, 대전
  // 2, 구미
  // 3, 서울
}

// for... of
for (region of ssafy) {
  console.log(region) // 광주, 대전, 구미, 서울
}

// forEach
ssafy.forEach((region, i) => {
  console.log(i, region)
  // 0, 광주
  // 1, 대전
  // 2, 구미
  // 3, 서울
})
```

방식	특징	비고
for loop	<ul style="list-style-type: none"><li>- <u>모든 브라우저 환경에서 지원</u></li><li>- 인덱스를 활용하여 배열의 요소에 접근</li><li>- break, continue 사용 가능</li></ul>	
for... of	<ul style="list-style-type: none"><li>- <u>일부 오래된 브라우저 환경에서 지원 X</u></li><li>- 인덱스 없이 배열의 요소에 바로 접근 가능</li><li>- break, continue 사용 가능</li></ul>	
forEach	<ul style="list-style-type: none"><li>- <u>대부분의 브라우저 환경에서 지원</u></li><li>- break, continue 사용 <b>불가능</b></li></ul>	<u><a href="#">Airbnb Style Guide 권장 방식</a></u>



## | 배열 (Arrays) Quiz

Q1. 배열은 원시 타입에 속하며 음수 인덱싱이 불가능하다는 특징이 있다. T/F

Q2. forEach 메서드는 배열의 각 요소들을 모아서 새로운 배열을 반환한다. T/F

Q3. 배열의 가장 마지막에 요소를 추가하는 메서드는 append이다. T/F

## 배열 (Arrays) Quiz

Q1. 배열은 원시 타입에 속하며 음수 인덱싱이 불가능하다는 특징이 있다.

F

A1. 자바스크립트의 배열은 참조 타입에 속한다.

Q2. forEach 메서드는 배열의 각 요소들을 모아서 새로운 배열을 반환한다.

F

A2. 배열의 각 요소들을 모아서 새로운 배열을 반환하는 메서드는 map 메서드이다.

Q3. 배열의 가장 마지막에 요소를 추가하는 메서드는 append이다.

F

A3. 배열의 가장 마지막에 요소를 추가하는 메서드는 push이다.

# 객체 (Objects)

## 객체의 정의와 특징

```
const me = {  
  name: '홍길동',           // key가 한 단어일 때  
  'phone number': '01012345678', // key가 여러 단어일 때  
  samsungProducts: {  
    buds: 'Galaxy Buds Pro',  
    galaxy: 'Galaxy s21',  
  },  
}  
  
console.log(me.name)           // 홍길동  
console.log(me['phone number']) // 01012345678  
console.log(me.samsungProducts) // {buds: "Galaxy Buds Pro" ...  
console.log(me.samsungProducts.buds) // Galaxy Buds Pro
```

- 객체는 속성(property)의 집합이며 중괄호 내부에 key와 value의 쌍으로 표현
- key는 문자열 타입\*만 가능
  - (참고) key 이름에 띄어쓰기 등의 구분자가 있을 경우 따옴표로 묶어서 표현
- value는 모든 타입 가능
- 객체 요소 접근은 점 또는 대괄호로 가능
  - (참고) key 이름에 띄어쓰기 같은 구분자가 있을 경우 대괄호 접근만 가능

## | 객체 관련 ES6 문법 익히기

- ES6에 새로 도입된 문법들로 객체 생성 및 조작에 유용하게 사용 가능
  - 속성명 축약
  - 메서드명 축약
  - 계산된 속성명 사용하기
  - 구조 분해 할당\*
    - (참고) 구조 분해 할당은 [배열도 가능함](#)

## 객체 관련 ES6 문법 (1) – 속성명 축약 (shorthand)

□ 객체를 정의할 때 key와 할당하는 변수의 이름이 같으면 예시와 같이 축약 가능

□ 예시)

```
let books = ['Learning JS', 'Eloquent JS']
let magazines = null

// ES5
var bookShop = {
  books: books,
  magazines: magazines,
}
console.log(bookShop.books) // ['Learning JS', 'Eloquent JS']
```

```
let books = ['Learning JS', 'Eloquent JS']
let magazines = null

// ES6+ (축약 문법 사용)
var bookShop = {
  books,
  magazines,
}
console.log(bookShop.books) // ['Learning JS', 'Eloquent JS']
```

## 객체 관련 ES6 문법 (2) – 메서드명 축약 (shorthand)

□ 메서드\* 선언 시 function 키워드 생략 가능

○ 메서드\*: 어떤 객체의 속성이 참조하는 함수

□ 예시)

```
// ES5
var obj = {
  greeting: function () {
    console.log('Hi!')
  }
};

obj.greeting() // Hi!
```

```
// ES6+ (축약 문법 사용)
const newObj = {
  greeting() {
    console.log('Hi!')
  }
};

newObj.greeting() // Hi!
```

## 객체 관련 ES6 문법 (3) – 계산된 속성 (computed property name)

- 객체를 정의할 때 key의 이름을 표현식을 이용하여 동적으로 생성 가능
- 예시)

```
const key = 'regions'
const value = ['광주', '대전', '구미', '서울']

const ssafy = {
  [key]: value,
}

console.log(ssafy)           // { regions: Array(4) }
console.log(ssafy.regions)  // ["광주", "대전", "구미", "서울"]
```



## 객체 관련 ES6 문법 (4) – 구조 분해 할당 (destructuring assignment)

- 배열 또는 객체를 분해하여 속성을 변수에 쉽게 할당할 수 있는 문법
- 예시)

```
const userInformation = {
  name: 'ssafy kim',
  userId: 'ssafyStudent1234',
  phoneNumber: '010-1234-1234',
  email: 'ssafy@ssafy.com'
}

const name = userInformation.name
const userId = userInformation.userId
const phoneNumber = userInformation.phoneNumber
const email = userInformation.email
```

```
const userInformation = {
  name: 'ssafy kim',
  userId: 'ssafyStudent1234',
  phoneNumber: '010-1234-1234',
  email: 'ssafy@ssafy.com'
}

const { name } = userInformation
const { userId } = userInformation
const { phoneNumber } = userInformation
const { email } = userInformation

// 여러개도 가능
const { name, userId } = userInformation
```

## JSON (JavaScript Object Notation)

- key-value쌍의 형태로 데이터를 표기하는 언어 독립적 표준 포맷
- 자바스크립트의 객체와 유사하게 생겼으나 실제로는 문자열 타입
  - 따라서 JS의 객체로써 조작하기 위해서는 구문 분석(parsing)이 필수
- 자바스크립트에서는 JSON을 조작하기 위한 두 가지 내장 메서드 제공
  - JSON.parse()
    - JSON => 자바스크립트 객체
  - JSON.stringify()
    - 자바스크립트 객체 => JSON

## JSON (JavaScript Object Notation) 예시

```
// Object => JSON

const jsonData = JSON.stringify({
  coffee: 'Americano',
  iceCream: 'Cookie and cream',
})

console.log(jsonData)          // '{"coffee":"Americano", ...
console.log(typeof jsonData)  // string
```

```
// JSON => Object

const jsonData = JSON.stringify({
  coffee: 'Americano',
  iceCream: 'Cookie and cream',
})

const parsedData = JSON.parse(jsonData)

console.log(parsedData)        // {coffee: "Americano", ...
console.log(typeof parsedData) // object
```

## | 객체 실습

- 목표: 자바스크립트 객체 연습 (08-objects.js)
- 문제: 파일에 작성된 주석 참고

## | 객체 (Objects) Quiz

Q1. 객체의 key 값은 원시 타입만 사용 가능하다.

T/F

Q2. 객체 정의 시 key 값과 key에 할당하는 변수의 이름이 같으면 축약 가능하다.

T/F

Q3. JSON 데이터는 문자열 타입으로 표현된다.

T/F

## | 객체 (Objects) Quiz

Q1. 객체의 key값은 원시 타입만 사용 가능하다.

F

A1. 객체의 key값은 문자열 타입만 사용 가능하다.

Q2. 객체 정의 시 key 값과 key에 할당하는 변수의 이름이 같으면 축약 가능하다.

T

Q3. JSON 데이터는 문자열 타입으로 표현된다.

T