



## 2주차 : es6, 일급객체로서의 함수, Map과 Set



### [학습 목표]

1. 자바스크립트 es6 문법에서 새로 추가된 문법에 대해 학습해요.
2. 일급객체로서의 함수가 어떤 의미인지, 왜 중요한지 알며 실습을 통해 활용능력을 갖출 수 있어요.
3. Map과 Set에 대한 개념에 대해 학습하고, 각각에 대한 활용능력을 갖출 수 있어요.

### 1. 각종 es6 문법 소개

ECMAScript 6 (ES6)는 JavaScript의 버전 중 하나로, 2015년에 발표되었어요. ES6는 이전 버전인 ES5에서 새로운 문법과 기능을 도입하여 JavaScript 개발자들이 보다 쉽고 효율적으로 코드를 작성할 수 있도록 개선하였습니다.

2015년도에 대규모 문법적 향상 및 변경이 있었기 때문에, ES6가 항상 언급이 되곤 해요! 🔥

이미 배우신 문법도 있고, 배우지 않으신 문법도 있지만 여기에 '정리' 한다는 생각으로 넣었어요.

참고해주세요 🙏🙏🙏

#### ▼ (1) let, const



1주차에서 배웠지만, ES6에 포함되었다는 의미에서 다시한번 소개해 드립니다. 반복해서 나쁠 것은 없겠죠?! 한번 더 보자구요!! 이미 많이 익숙하신 분들은 skip, skip!! 👍

기존에 변수 선언을 위해 존재하던 `var`를 대체해서 나온 변수 선언에 사용 되는 키워드예요.

선언과 할당이 무엇이었는데 중요하다고 했었죠? `var`, `const`, `let`의 차이를 알 수 있기 위해서는 선언과 할당의 의미를 아주 정확히 알고 있어야 하기 때문이죠.

- 선언: 변수명을 자바스크립트 엔진에 알리는 것이예요.

- 할당: 변수에 값을 저장하는 것 (= 할당연산자)이에요.

#### let 과 const 의 특징

- **let**: 재할당은 가능하고, 재선언은 불가능해요.

```
let value = "value1"
console.log(value) // value1

value = "value2" // 재할당 가능
console.log(value) // value2

let value = "value3" // 재선언 불가능, SyntaxError: Identifier 'value' has already been declared
```

- **const**: 재할당, 재선언이 불가능, 초기값이 없으면 선언 불가능해요.

```
const value; // 초기값 없이 선언 불가능, SyntaxError: Missing initializer in const declaration
---
const value = "value1"
console.log(value) // value1

value = "value2" // 재할당 불가능, TypeError: Assignment to constant variable.

const value = "value2" // 재선언 불가능, SyntaxError: Identifier 'value' has already been declared
```

#### ▼ 잠깐 알아보는 var의 특징

1. var는 재할당, 재선언이 가능해요.

```
var name = "name1"
console.log(name) // name1

var name = "name2"
console.log(name) // name2
```

2. var는 호이스팅됩니다(3주차에 자세히! 지금은 몰라도 됩니다 🤔)

```
console.log(name) // undefined

var name = "name1"
```

## ▼ (2) 화살표 함수 (Arrow Function)

**function** 이나 **return** 키워드 없이 함수를 만드는 방법이에요. 앞선 함수 만드는 방법 시간에 설명드렸었죠! 이것도 역시 **ES6에서** 처음 소개된 개념입니다.

```
// ES5
function func() {
```

```

    return true
  }

  //ES6
  const func = () => true
  const func = () => {
    return true
  }

  () => {}
  parm => {}
  (parm1, parm2, ...parms) -> {}

  // 익명 화살표 함수
  () => {}

```

## this

`function` 은 호출을 할 때 `this` 가 정해지지만, 화살표 함수는 선언할 때 `this` 가 정해진다.

### ▼ (3) 삼항 연산자 (ternary operator)

```

condition ? expr1 : expr2

console.log(true ? "참" : "거짓") // 참
console.log(false ? "참" : "거짓") // 거짓

```

### ▼ (4) 구조 분해 할당 (Destructuring)

구조 분해 할당이란?

배열 `[]` 이나 객체 `{}` 의 속성을 분해해서 그 값을 변수에 담을 수 있게 해주는 문법이에요.

```

// 배열의 경우
let [value1, value2] = [1, "new"];
console.log(value1); // 1
console.log(value2); // "new"

let arr = ["value1", "value2", "value3"];
let [a,b,c] = arr;
console.log(a,b,c) // value1 value2 value3

// let [a,b,c] = arr; 은 아래와 동일!
// let a = arr[0];
// let b = arr[1];
// let c = arr[2];

let [a,b,c,d] = arr
console.log(d) // undefined

let [a,b,c,d = 4] = arr
console.log(d) // 4

```

```
// 객체의 경우
let user = {name: "nbc", age: 30};
let {name, age} = user;

// let name = user.name;
// let age = user.age;

console.log(name, age) // nbc 30

// 새로운 이름으로 할당
let {name: newName, age: newAge} = user;
console.log(name, age) // ReferenceError: name is not defined
console.log(newName, newAge) //nbc 30

let {name, age, birthDay} = user;
console.log(birthDay) // undefined

let {name, age, birthDay = "today"} = user;
console.log(birthDay) // today
```

## ▼ (5) 단축 속성명 (property shorthand)

객체의 key와 value 값이 같다면, 생략 가능해요.

```
const name = "nbc"
const age = "30"

const obj = {
  name,
  age: newAge
}

const obj = {
  name,
  age
}
```

## ▼ (6) 전개 구문 (Spread)

배열이나 객체를 전개하는 문법이에요. 구조분해할당과 함께 정말 많이 사용됩니다.

```
// 배열
let arr = [1,2,3];

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

// 객체
let user = {name: "nbc", age: 30};
let user2 = {...user}

user2.name = "nbc2"

console.log(user.name) // nbc
console.log(user2.name) // nbc2
```

## ▼ (7) 나머지 매개변수(rest parameter)

```
function func (a, b, ...args) {  
  console.log(...args)  
}  
  
func(1, 2, 3) // 3  
func(1, 2, 3, 4, 5, 6, 7) // 3 4 5 6 7
```

## ▼ (8) 템플릿 리터럴 (Template literals)


여러 줄로 이뤄진 문자열과 문자 보간기능을 사용하게 만들어 주는 문자열 리터럴 표현식이에요.


백틱(`) 과 \${} 로 표현합니다.

### 공식문서 링크(아래)

#### Template literals - JavaScript | MDN

템플릿 리터럴은 내장된 표현식을 허용하는 문자열 리터럴입니다. 여러 줄로 이뤄진 문자열과 문자 보간기능을 사용할 수 있습니다. 이전 버전의 ES2015 사양 명세에서는 "template strings" (템플릿 문자열) 라고 불러 왔습니다.

 [https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template\\_literals](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Template_literals)

 mdn web docs

```
"string"  
'string'  
new String("string")  
  
`string text`  
  
`string text  
string text line2`  
  
`string text ${value} text`
```

## ▼ (9) named export vs default export

### 1. default Export

```
// name.js  
const Name = () => {  
}  
  
export default Name  
  
// other file  
// 아무 이름으로 import 가능  
import newName from "name.js"  
import NameFromOtherModule from "name.js"
```

## 2. named export

```
// 하나의 파일에서 여러 변수/클래스 등을 export 하는 것이 가능

export const Name1 = () => {
}

export const Name2 = () => {
}

// other file
import {Name1, Name2} from "name.js"
import {newName} from "name.js" // x

// 다른 이름으로 바꾸려면 as 사용
import {Name1 as newName, Name2} from "name.js"

// default export 처럼 가져오려면 * 사용
import * as NameModule from "name.js"

console.log(NameModule.Name1)
```

## 2. 일급 객체로서의 함수

자바스크립트에서 함수는 일급 객체(First-Class Object)라고 해요. 그래서 함수를 객체처럼 여러가지 방식으로 다룰 수 있어요. 일반 객체와 달리 함수는 특별한 능력을 가지고 있습니다.

**일급객체(First-class Object)란 다른 객체들에 일반적으로 적용 가능한 연산을 모두 지원하는 객체를 가리킨다. [위키백과]**

함수가 일급 객체로 취급되기 때문에, 우리는 함수를 매우 유연하게 사용할 수 있어요. 그래서 자바스크립트에서 함수는 **매우 중요한 개념(이렇게 따로 빼야 할 만큼)**이에요. 아래에서 함수가 일급 객체로 취급되는 5가지 경우에 대해 알아보게요.

### ▼ (1) 변수에 함수를 할당

함수는 변수에 할당할 수 있습니다. 함수는 값으로 취급되기 때문에, 다른 변수와 마찬가지로 변수에 할당할 수 있습니다. 변수에 할당된 함수는 나중에 사용할 수 있습니다.

```
const sayHello = function() {
  console.log('Hello!');
};

sayHello(); // "Hello!" 출력
```

### ▼ (2) 함수를 인자로 다른 함수에 전달

함수는 다른 함수에 인자로 전달될 수 있습니다. 함수가 값으로 취급되기 때문에, 다른 함수의 인자로 전달할 수 있습니다. 이것은 콜백(callback)이나 고차 함수(higher-order function)를 작성하

는 데 사용됩니다.



### 콜백 함수. 그리고, 고차 함수(Higher-Order Function)란?

- 콜백 함수는 어떠한 함수의 매개변수로 쓰이는 함수를 말해요. 4주차 때 정——말 깊게 배운답니다 😎
- 고차 함수는 함수를 인자로 받거나 함수를 출력으로 반환하는 함수를 말해요! 함수를 다루는 함수라고도 하죠. 정리하면 콜백함수는 고차함수라고도 할 수 있겠네요.

```
function callFunction(func) {  
  func();  
}  
  
const sayHello = function() {  
  console.log('Hello!');  
};  
  
callFunction(sayHello); // "Hello!" 출력
```

### ▼ (3) 함수를 반환

함수는 다른 함수에서 반환될 수 있습니다. 함수는 값으로 취급되기 때문에, 다른 함수에서 반환할 수 있습니다. 이것은 함수 팩토리(factory)나 클로저(closure)를 작성하는 데 사용됩니다.

```
function createAdder(num) {  
  return function(x) {  
    return x + num;  
  }  
}  
  
const addFive = createAdder(5);  
console.log(addFive(10)); // 15 출력
```

### ▼ (4) 객체의 프로퍼티로 함수를 할당

함수는 객체의 프로퍼티로 할당될 수 있습니다. 객체의 메소드로 함수를 호출할 수 있습니다.

```
const person = {  
  name: 'John',  
  sayHello: function() {  
    console.log(`Hello, my name is ${this.name}`);  
  }  
};  
  
person.sayHello(); // "Hello, my name is John" 출력
```

### ▼ (5) 배열의 요소로 함수를 할당

함수는 배열의 요소로 할당될 수 있습니다. 이것은 함수를 배열에서 사용할 수 있게 합니다.

```
const myArray = [
  function(a, b) {
    return a + b;
  },
  function(a, b) {
    return a - b;
  }
];

console.log(myArray[0](5, 10)); // 15 출력
console.log(myArray[1](10, 5)); // 5 출력
```

함수가 일급 객체로 취급되기 때문에, 자바스크립트에서 함수는 매우 유연하게 사용될 수 있습니다. 이것은 등 다양한 프로그래밍 패턴에서 사용됩니다. 함수를 일급 객체로 다룰 수 있다는 것은, 코드를 더 간결하고 모듈화된 형태로 작성할 수 있게 해줍니다.

함수를 일급 객체로 다룰 수 있다는 것은, 함수를 다양하게 조합할 수 있다는 것을 의미합니다. 새로운 함수를 반환하는 함수를 작성하면, 함수를 조합하여 더 복잡한 기능을 구현할 수 있어요. 이것을 활용하면 코드를 더욱 간결하게 작성할 수 있으며, 유지 보수도 쉬워집니다.

```
function multiplyBy(num) {
  return function(x) {
    return x * num;
  }
}

function add(x, y) {
  return x + y;
}

const multiplyByTwo = multiplyBy(2);
const multiplyByThree = multiplyBy(3);

const result = add(multiplyByTwo(5), multiplyByThree(10)); // 35 출력
```

이처럼 함수가 일급 객체로 취급되는 것은 자바스크립트에서 함수를 다양한 방식으로 사용할 수 있게 해줍니다. 함수를 객체나 배열과 같은 일반적인 자료형과 동일한 방식으로 사용할 수 있기 때문에, 코드를 더 간결하고 모듈화된 형태로 작성할 수 있습니다.

### 3. Map과 Set

JavaScript에서 **객체(object)**와 **배열(array)**을 이용하면 굉장히 다양하고 복잡한 프로그래밍을 할 수 있습니다. 그럼에도 불구하고 여전히 현실세계의 여러가지 문제들을 **‘프로그래밍’적으로** 반영하기엔 많이 부족해요. **Map**과 **Set**은 이러한 한계를 극복하고자 비교적 최근 등장한 자료구조라고 할 수 있습니다.



이 두 자료 구조는 데이터의 구성, 검색 및 사용을 객체나 배열보다 효율적으로 처리할 수 있기 때문에 많이 각광받고 있어요. 이제 JavaScript에서 `Map` 과 `Set` 을 사용하는 방법을 살펴보겠습니다 🚀🚀

## ▼ (1) Map

**Map**은 키-값 쌍을 저장하는 객체와 비슷합니다. **Map**은 각 쌍의 키와 값을 저장하며, 객체와 달리 키로 사용할 수 있는 모든 유형을 사용할 수 있습니다. **Map**은 키가 정렬된 순서로 저장되기 때문에, 추가한 순서대로 반복할 필요가 없습니다. **Map**을 사용하면 다음과 같은 작업을 수행할 수 있습니다.

- 키-값 쌍 추가 및 검색(set)
- 키-값 쌍 삭제(delete)
- 모든 키-값 쌍 제거(clear)
- **Map** 크기 및 존재 여부 확인(size)



[맵에는 다음과 같은 주요 메서드와 프로퍼티가 있습니다]

- `new Map()` – 맵을 만듭니다.
- `map.set(key, value)` – `key` 를 이용해 `value` 를 저장합니다.
- `map.get(key)` – `key` 에 해당하는 값을 반환합니다. `key` 가 존재하지 않으면 `undefined` 를 반환합니다.
- `map.has(key)` – `key` 가 존재하면 `true`, 존재하지 않으면 `false` 를 반환합니다.
- `map.delete(key)` – `key` 에 해당하는 값을 삭제합니다.
- `map.clear()` – 맵 안의 모든 요소를 제거합니다.
- `map.size` – 요소의 개수를 반환합니다.

위 내용을 바탕으로 실습을 진행해볼까요?

### <Map 생성 및 사용>

새로운 **Map**을 만들려면 **Map()** 생성자를 사용합니다.

```
const myMap = new Map();
```

이제 **Map**에 값을 추가하려면 **set()** 메소드를 사용합니다.

```
myMap.set('key', 'value');
```

**Map**에서 값을 검색하려면 **get()** 메소드를 사용합니다.

```
console.log(myMap.get('key')); // 'value' 출력
```

## <Map의 반복>

Map에서는 `keys()`, `values()`, `entries()` 메소드를 사용하여 키, 값 및 키-값 쌍을 반복할 수 있습니다.



### [for ...of 반복문]

`for of` 반복문은 ES6에 추가된 새로운 컬렉션 전용 반복 구문입니다. `for of` 구문을 사용하기 위해선 컬렉션 객체가 `[Symbol.iterator]` 속성을 가지고 있어야만 합니다(직접 명시 가능).

```
var iterable = [10, 20, 30];

for (var value of iterable) {
  console.log(value); // 10, 20, 30
}
```

영어사전 단어·속어 1-5 / 6건

[iterator](#)

반복자

iterator는 반복자라는 말이에요. 요소 하나하나를 반복할 수 있도록 배열 또는 객체와 비슷한 형태로 열거되어있는 자료구조로 이해해주시면 돼요. 아래 예시 코드에서

`myMap.keys()` 으로 쓸 수 있는 이유는 `myMap.key()` 가 반환하는 값이 iterator이기 때문이에요.

```
const myMap = new Map();
myMap.set('one', 1);
myMap.set('two', 2);
myMap.set('three', 3);

for (const key of myMap.keys()) {
  console.log(key);
}

for (const value of myMap.values()) {
  console.log(value);
}

for (const entry of myMap.entries()) {
  console.log(`${entry[0]}: ${entry[1]}`);
}
```

## <Map의 크기 및 존재 여부 확인>

Map의 크기를 확인하려면 **size** 속성을 사용합니다.

```
console.log(myMap.size); // 3 출력
```

특정 키가 Map에 존재하는지 여부를 확인하려면 **has()** 메소드를 사용합니다.

```
console.log(myMap.has('two')); // true 출력
```

## ▼ (2) Set

Set은 고유한 값을 저장하는 자료 구조입니다. Set은 값만 저장하며, 키를 저장하지 않습니다. Set은 값이 중복되지 않는 유일한 요소로만 구성됩니다. Set을 사용하면 다음과 같은 작업을 수행할 수 있습니다.

- 값 추가 및 검색
- 값 삭제
- 모든 값 제거
- Set 크기 및 존재 여부 확인

## <Set 생성 및 사용>

새로운 Set을 만들려면 **Set()** 생성자를 사용합니다.

```
const mySet = new Set();
```

이제 Set에 값을 추가하려면 **add()** 메소드를 사용합니다.

```
mySet.add('value1');  
mySet.add('value2');
```

Set에서 값을 검색하려면 **has()** 메소드를 사용합니다.

```
console.log(mySet.has('value1')); // true 출력
```

## <Set의 반복>

Set에서는 **values()** 메소드를 사용하여 값을 반복할 수 있습니다.

```
const mySet = new Set();  
mySet.add('value1');  
mySet.add('value2');
```

```
mySet.add('value3');

for (const value of mySet.values()) {
  console.log(value);
}
```

## < Set 의 크기 및 존재 여부 확인 >

Set 의 크기를 확인하려면 size 속성을 사용합니다.

```
console.log(mySet.size); // 3 출력
```

특정 값을 Set 에서 검색하여 존재하는지 여부를 확인하려면 has() 메소드를 사용합니다.

```
console.log(mySet.has('value2')); // true 출력
```

JavaScript에서 Map 과 Set 은 두 가지 다른 유형의 자료 구조입니다. Map 은 키-값 쌍을 저장하는 객체와 비슷하며, Set 은 고유한 값을 저장하는 자료 구조입니다. Map 및 Set 은 모두 값 추가, 검색, 삭제 및 모든 값 제거를 수행할 수 있습니다. Map 및 Set 을 사용하여 효율적인 데이터 구성 및 검색을 수행할 수 있습니다.

## 4. 숙제

### ▼ 문제 설명

문자열로 구성된 리스트 strings와, 정수 n이 주어졌을 때, 각 문자열의 인덱스 n번째 글자를 기준으로 오름차순 정렬하려 합니다. 예를 들어 strings가 ["sun", "bed", "car"]이고 n이 1이면 각 단어의 인덱스 1의 문자 "u", "e", "a"로 strings를 정렬합니다.

### 제한 조건

- strings는 길이 1 이상, 50이하인 배열입니다.
- strings의 원소는 소문자 알파벳으로 이루어져 있습니다.
- strings의 원소는 길이 1 이상, 100이하인 문자열입니다.
- 모든 strings의 원소의 길이는 n보다 큼니다.
- 인덱스 1의 문자가 같은 문자열이 여럿 일 경우, 사전순으로 앞선 문자열이 앞쪽에 위치합니다.

### 입출력 예

strings	n	return
["sun", "bed", "car"]	1	["car", "bed", "sun"]

["abce", "abcd", "cdx"] 2

["abcd", "abce", "cdx"]

## 입출력 예 설명

**입출력 예 1** "sun", "bed", "car"의 1번째 인덱스 값은 각각 "u", "e", "a" 입니다. 이를 기준으로 strings를 정렬하면 ["car", "bed", "sun"] 입니다.

**입출력 예 2** "abce"와 "abcd", "cdx"의 2번째 인덱스 값은 "c", "c", "x"입니다. 따라서 정렬 후에는 "cdx"가 가장 뒤에 위치합니다. "abce"와 "abcd"는 사전순으로 정렬하면 "abcd"가 우선하므로, 답은 ["abcd", "abce", "cdx"] 입니다.

### ▼ 문제 풀이 팁

#### 1) 입력으로 받아온 문자열의 '배열'을 정렬하는 문제입니다.

우리의 과제는 정렬하는 로직을 구현하는 것 입니다.

#### 2) 정렬하는 로직을 구현하는 아이디어 입니다.

1. 기본적으로 사전식 정렬입니다. 문자열 배열같은 경우 sort로 정렬이 가능합니다.
2. 그런데 인덱스에 해당하는 문자 순서로 정렬하고, 그 인덱스에 해당하는 문자가 같은 경우 사전식으로 정렬해주는 작업이 필요합니다.
3. 물론 성능적인 측면에서 더 좋은 방법이 있지만, 지금은 가장 쉬운 방법을 채택하려고 합니다.

#### 3) 결론적으로 아래와 같은 순서로 구현하시면 쉽습니다.

1. 문자열 앞에 인덱스에 해당하는 문자를 붙인다
  - a. ["sun", "bed", "car"], 1 이라면 → ["usun", "ebed", "acar"]
2. 사전순으로 정렬한다
  - a. ["acar", "ebed", "usun"]
3. 정렬된 배열의 가장 앞 글자를 떼낸다
  - a. ["car", "bed", "sun"]

4) 위와 같은 방법으로 구현하셨다면, 성능적으로(시간복잡도 상) 개선을 해봐도 좋지만, 오늘 배운 배열의 메서드를 이용해서 1,2,3번의 로직을 구현하는 것을 연습해보시는걸 추천합니다 😊

### ▼ 정답 코드 / 해설 영상

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/1ae6f1f2-f294-42fd-a433-53d567a5ccb0/2%E1%84%8C%E1%85%AE%E1%84%8E%E1%85%A1.mp4>

```
function solution(strings, n) {  
  let result = [];
```

```
// 문자열 가장앞 글자 붙인 문자 배열 만들기
for (let i = 0; i < strings.length; i++) {
  strings[i] = strings[i][n] + strings[i];
}

// 문자열 사전순 정렬
strings.sort();

// 앞글자 제거 후 리턴
for(let j = 0; j < strings.length; j ++) {
  strings[j] = strings[j].replace(strings[j][0], "");
  result.push(strings[j]);
}

return result;
}
```