JavaScript

◆ ES6 문법

정수아

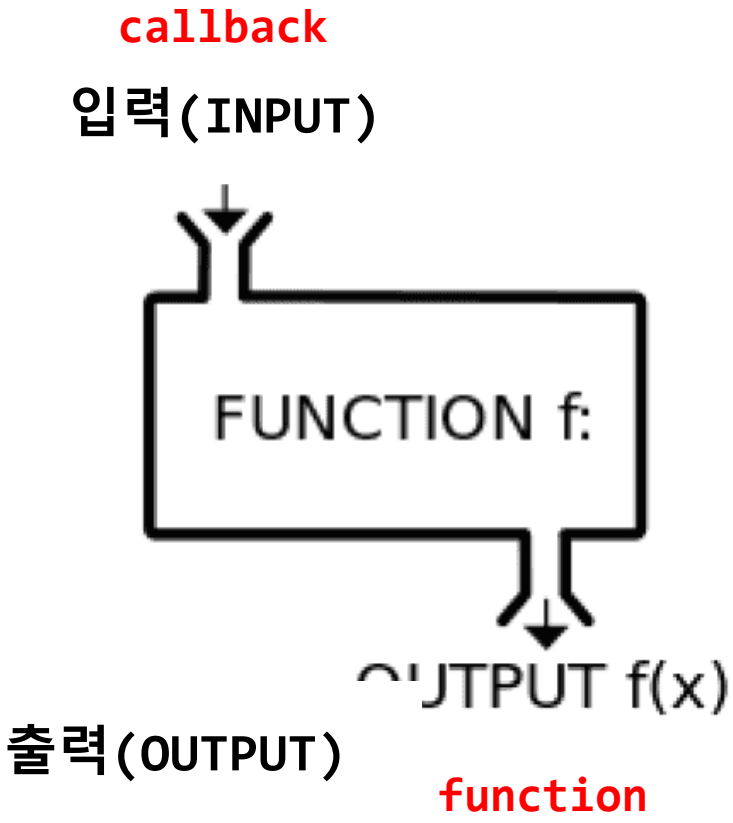
Contents

01 고차함수란?

고차함수(Higher-order function)

❖ **고차함수**

- 인자로 함수를 받거나 함수를 반환하는 함수
- 함수 안에서 다른 함수를 원하는 시점에 호출할 수 있는 권한을 가진 함수



고차함수(Higher-order function)

❖ 콜백함수

- 인자로 전달되는 시점에 바로 함수를 호출하는 것이 아닌 고차 함수 내부에서 필요한 순간에 호출을 하는 함수

```
const add = (a, b) => a + b;
const sub = (a, b) => a - b;

function calculator(a, b, calc) {
   let result = calc(a, b);
   console.log(result);
   return result;
}

calculator(1, 1, add);
calculator(1, 2, sub);
```

[실습] 콜백함수

❖ 일반 함수

```
setTimeout(function() {
  console.log("5초 후에 실행됩니다.");
}, 5000);
```

❖ 화살표 함수

```
setTimeout(() => {
  console.log('5초 후에 실행됩니다.');
}, 5000);
```

02

고차함수 종류

forEach()

❖ forEach()를 이용한 배열 순회

- 콜백 함수를 배열의 각 요소에 대해 한 번씩 실행

```
const numbers = [10, 20, 30, 40, 50];

// for문을 이용한 배열 순회
for(i = 0; i < numbers.length; i++){
   console.log(numbers[i]);
}

// forEach를 이용한 배열 순회
numbers.forEach(function(number){
   console.log(number)
});

// 화살표 함수를 이용
numbers.forEach(number => console.log(number));
```

forEach()

❖ **forEach(currentValue, index)**

- currentValue
  - 현재 요소
- index
  - 현재 요소의 인덱스

```
numbers.forEach(function(number, index) {
   console.log("Index : " + index + " Value : " + number);
});
```

- 실행 결과

```
Index : 0 Value : 10
Index : 1 Value : 20
Index : 2 Value : 30
Index : 3 Value : 40
Index : 4 Value : 50
```

forEach()

❖ **forEach(currentValue, index, array)**

- array
  - forEach()를 호출한 배열

```
numbers.forEach(function(number, index, array) {
   console.log(array);
});
```

- 실행 결과

```
[ 10, 20, 30, 40, 50 ]
[ 10, 20, 30, 40, 50 ]
[ 10, 20, 30, 40, 50 ]
[ 10, 20, 30, 40, 50 ]
[ 10, 20, 30, 40, 50 ]
```

find()

❖ find()

- 조건을 만족하는 첫 번째 아이템을 찾아서 반환해줌

```
const fruits = ["apple", "banana", "melon", "orange"];
```

```
// function
const favorite1 = fruits.find(function(name) {
   return name === "melon";
});

console.log(favorite1);
```

```
// 화살표 함수
const favorite2 = fruits.find((name) => name === "melon");
console.log(favorite2);
```

findIndex()

❖ findIndex()

- 조건을 만족하는 첫 번째 아이템의 인덱스를 반환

```
// function
fIndex1 = fruits.findIndex(function (name) {
  return name === "melon";
});
console.log(fIndex1);
```

```
// 화살표 함수
fIndex2 = fruits.findIndex((name) => name === "melon");
console.log(fIndex2);
```

- 실행 결과

```
2
```

some()

❖ some()

- 배열의 아이템들이 조건을 부분적으로 만족하는지 확인

```
// function
const some1 = fruits.some(function (name) {
   return name === "melon";
});

console.log(some1);
```

```
// 화살표 함수
const some2 = fruits.some((name) => name === "melon");
console.log(some2);
```

- 실행 결과

```
true
```

every()

❖ every()

- 배열의 아이템들이 조건을 전부 만족하는지 확인

```
// function
const every1 = fruits.every(function (name) {
   return name === "melon";
});
console.log(every2);
```

```
// 화살표 함수
const every2 = fruits.every((name) => name === "melon");
console.log(every2);
```

- 실행 결과

```
false
```

filter()

❖ filter()

- 조건에 맞는 모든 아이템들을 새로운 배열로 생성

```
const fruits = ["apple", "banana", "melon", "orange", "melon"];
```

```
// function
const filter1 = fruits.filter(function (name) {
   return name === "melon";
});
console.log(filter1);
```

```
// 화살표 함수
const filter2 = fruits.filter((name) => name === "melon");
console.log(filter2);
```

- 실행 결과

```
['melon', 'melon']
```

Map()

❖ Map()

- 배열 내 모든 아이템에 콜백함수를 적용한 결과를 모아 새로운 배열을 반환

```
const array = [10, 20, 30, 40, 50];
```

```
// function
const map1 = array.map(function(item) {
   return item * 2;
});
console.log(map1);
```

```
// 화살표 함수
const map2 = array.map((item) => item * 2);
console.log(map2);
```

- 실행 결과

```
[ 20, 40, 60, 80, 100 ]
```

Map()

❖ **Map(currentValue, index)**

- currentValue
  - 현재 요소
- index
  - 현재 요소의 인덱스

```
// function
const map1 = array.map(function (item, index) {
  return item * index;
});
console.log(map1);
```

```
// 화살표 함수
const map2 = array.map((item, index) => item * index);
console.log(map2);
```

- 실행 결과

```
[ 0, 20, 60, 120, 200 ]
```

03

스프레드 연산자

스프레드 연산자

❖ 스프레드 연산자

```
function add(a, b, c) {
    return a + b + c;
}

const nums = [10, 20, 30];
const result = add(nums[0], nums[1], nums[2]);

console.log(result);
```

- 실행 결과

```
60
```

스프레드 연산자

❖ 스프레드 연산자

```
function add(a, b, c) {
  return a + b + c;
}

const nums = [10, 20, 30];
const result = add(...nums);

console.log(result);
```

- 실행 결과

```
60
```

스프레드 연산자

❖ 함수와 스프레드 연산자

```
function rest(num1, num2, ...nums) {
    console.log(nums);
}

rest(10, 20, 30, 40, 50);
```

- 실행 결과

```
[30, 40, 50]
```

스프레드 연산자

❖ 배열과 스프레드 연산자

```
const num1 = [1, 2, 3];
const num2 = [4, 5, 6];

let newArr = num1.concat(num2);

console.log(newArr);
```

- 실행 결과

```
[ 1, 2, 3, 4, 5, 6 ]
```

스프레드 연산자

❖ 배열과 스프레드 연산자

```
const num1 = [1, 2, 3];
const num2 = [4, 5, 6];

const newArr = [...num1, ...num2];
console.log(newArr);
```

- 실행 결과

```
[ 1, 2, 3, 4, 5, 6 ]
```

스프레드 연산자

❖ 객체와 스프레드 연산자

```
const dog = { name: "바둑이", age: 5, color: "brown" };
const newDog = {
  ...dog,
  owner: "soo",
};
console.log(dog);
console.log(newDog);
```

- 실행 결과

```
{ name: '바둑이', age: 5, color: 'brown' }
{ name: '바둑이', age: 5, color: 'brown', owner: 'soo' }
```

04 구조 분해 할당

구조 분해 할당

❖ 구조 분해 할당

- 배열이나 객체의 속성을 해체하여 그 값을 개별 변수에 담을 수 있게 하는 표현식

```
let a, b, rest;
[a, b] = [10, 20];

console.log(a);          // 10
console.log(b);          // 20

[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(rest);       // [30,40,50]
```

```
const foo = ["one", "two", "three"];
const [red, yellow, green] = foo;

console.log(red);        // "one"
console.log(yellow);     // "two"
console.log(green);      // "three"
```

05 Map과 Set

Set 객체

❖ **Set 객체**
- 중복되지 않는 유일한 값들의 집합

❖ **특징**
- 동일한 값을 중복하여 포함할 수 없음
- 각각의 아이템은 순서가 없음
- 인덱스로 요소에 접근할 수 없음

[실습] Set 객체

```
const set1 = new Set();
console.log(set1); // Set(0) {}

const set2 = new Set([1, 2, 3, 4, 5]);
console.log(set2); // Set(5) {1, 2, 3, 4, 5}

const set3 = new Set("hello");
console.log(set3); // Set(4) {"h", "e", "l", "o"}
```

- 실행 결과

```
Set(0) {}
Set(5) {1, 2, 3, 4, 5}
Set(4) {"h", "e", "l", "o"}
```

[실습] Set 객체에 적용할 수 있는 함수

```
// set 생성
const set = new Set([10, 20, 30]);

// 사이즈 확인
console.log(set.size);           // 3

// 데이터 존재 여부
console.log(set.has(10));        // true
console.log(set.has(40));        // false

// 데이터 추가
set.add(40);
console.log(set);                // Set(4) { 10, 20, 30, 40 }

// 데이터 순회
set.forEach((item) => console.log(item)); // 10 20 30 40

// 삭제
set.delete(20);
console.log(set);                // Set(3) { 10, 30, 40 }


// 전부 삭제
set.clear();
console.log(set);                // Set(0) {}
```

Map 객체

❖ **Map 객체**
- 키와 값의 쌍으로 이루어진 컬렉션

❖ **객체와 Map 객체의 차이**
- 객체의 Key는 문자열만 가능
- Map 객체의 Key는 모든 자료형이 가능

Map 객체

❖ Map 생성 및 key, value 설정

```
// Map 생성
const map = new Map();

// set() : key와 value 설정
map.set("name", "soo");          // 문자형 키
map.set(1, "number");            // 숫자형 키
map.set(true, "trueValue");      // 불린형 키

console.log(map);
```

- 실행 결과

```
Map(3) { 'name' => 'soo', 1 => 'number', true => 'trueValue' }
```

Map 객체

❖ **Map은 key를 문자열로 변환하지 않음**

```
// get() : map에서 key에 해당하는 값 가져오기
console.log(map.get("name"));    // 'soo'
console.log(map.get(1));         // 'number'
```

❖ **Map의 요소 개수**

```
console.log(map.size);           // 3
```

[실습] Map 객체에 적용할 수 있는 함수

```
// Map 생성
const map = new Map();

// set() : key와 value 설정
map.set("name", "soo");
map.set("age", 20);
map.set("addr", "seoul");

// name 키 값 출력
console.log(map.get("name"));          // ‘soo’

// score 값 추가
map.set("score", 100);

// size 확인
console.log(map.size);                 // 4

// age 값 삭제
map.delete("age");

// map 확인
console.log(map);                      // Map(3) { 'name' => 'soo',
                                                    'addr' => 'seoul',
                                                    'score' => 100 }
```


THANK YOU