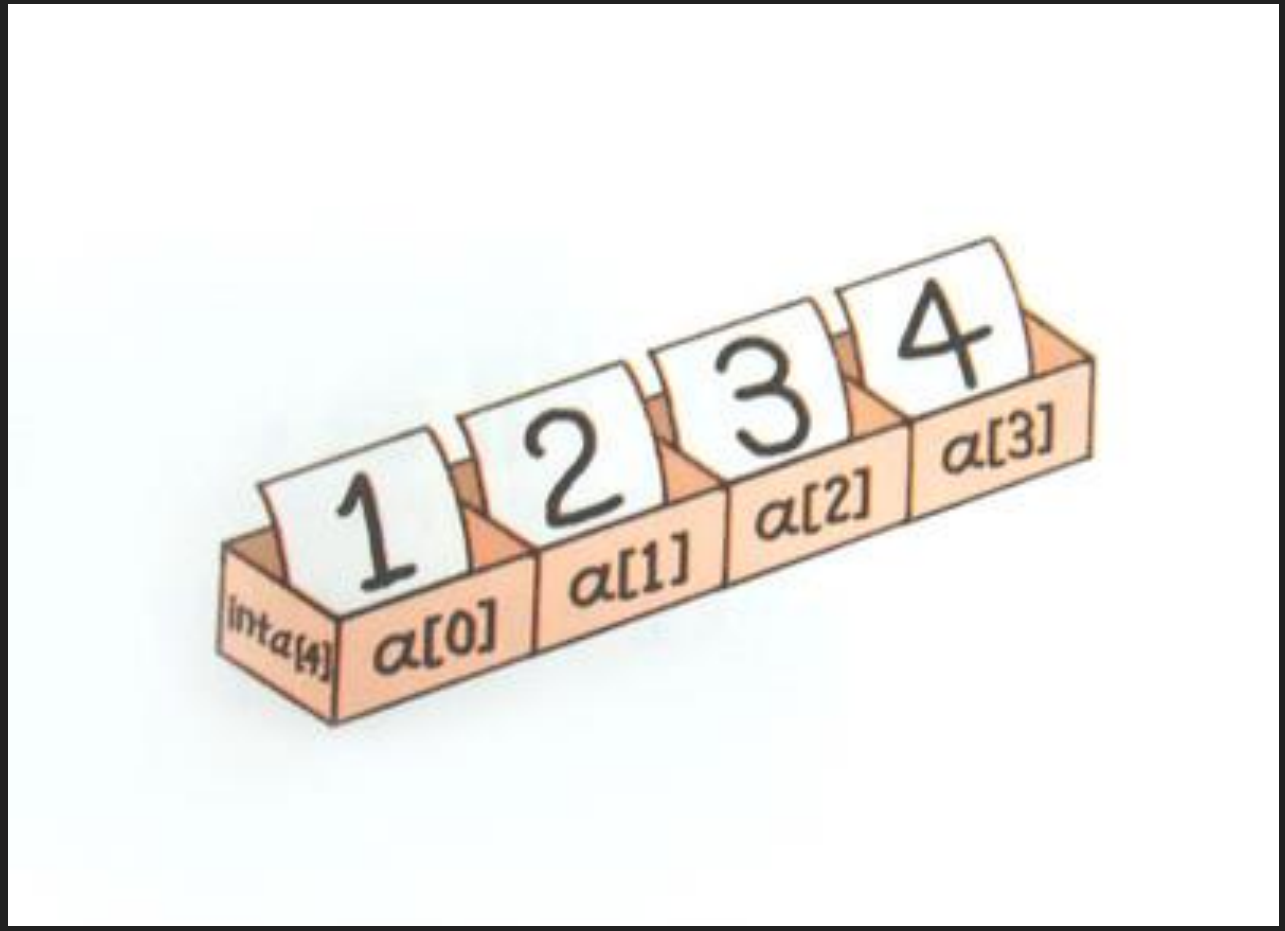
JavaScript

배열 – 김근형 강사

배열

- 배열



배열

- 배열
  - 배열은 인덱스를 가지는 복수의 자료를 저장할 수 있는 자료구조 이다.
  - 데이터는 순차적으로 저장되며 중복이 가능하다.
  - 저장된 데이터에는 인덱스를 이용하여 접근이 가능하다.
  - 배열의 선언 방법은 여러가지 방법이 있으며 대표적으로 new 명령을 사용하는 방법과 리터럴 표기법(literal notation) 을 사용하여 생성하는 방식이 있다

배열

- 배열 선언 방법

```
// 길이가 0인 예전 방식의 배열 생성
var myArray1 = new Array();

// 길이가 10인 예전 방식의 배열 생성
// 10개의 자리에는 아무것도 들어 있지 않습니다.
var myArray2 = new Array(10);

// 길이가 0인 literal notation 방식의 배열 생성
var myArray3 = [];

// 예전 방식의 생성과 동시에 초기화
var myArray4 = new Array(1, 2, 3,"홍길동", "아무개");

// literal notation 방식의 생성과 동시에 초기화
var myArray5 = [1, 2, 3, "홍길동", "아무개"];
```

배열

- 배열 요소 접근
  - 배열 요소 접근은 인덱스 번호를 통해 접근이 가능하다.
  - 인덱스 번호는 0부터 시작하며 "현재 배열의 최대 길이 – 1" 로 끝난다.
  - 현재 배열의 길이 이상의 인덱스를 가질 수 없으며 만약 잘못된 인덱스로 배열에 접근할 경우 undefined가 출력된다.

배열

- 배열 요소 접근

```
var arrNumber = new Array(); //배열선언

arrNumber[0] = 1;
arrNumber[1] = 2;
arrNumber[2] = 3;
arrNumber[3] = 4;
arrNumber[4] = 5;

console.log(arrNumber[0]);
console.log(arrNumber[1]);
console.log(arrNumber[2]);
console.log(arrNumber[3]);
console.log(arrNumber[4]);
```

```
1
2
3
4
5
```

Array.length

- Array.length
  - 현재 배열의 길이를 나타낼 경우 사용하는 프로퍼티.
  - 배열 리터럴을 넣게 되면 해당 length가 증가한다.
  - length 프로퍼티는 배열의 원소 개수를 나타내지만 실제로 배열에 존재하는 원소 개수와 일치하지는 않는다.

Array.length

- Array.length 예제

```
// 빈 배열
var emptyArr = [];
console.log(emptyArr[0]); // undefined

// 배열 요소 동적 생성
emptyArr[0] = 100;
emptyArr[3] = 'eight'
emptyArr[7] = true;
console.log(emptyArr); // [100, undefined × 2, "eight", undefined × 3, true]
console.log(emptyArr.length); // 8
```

```
undefined
▶ (8) [100, empty × 2, "eight", empty × 3, true]
8
```

Array.length

- Array.length 예제

```
var arr = [];
console.log(arr.length); // 0

arr[0] = 0; // arr.length = 1
arr[1] = 1; // arr.length = 2
arr[2] = 2; // arr.length = 3
arr[100] = 100;
console.log(arr.length); // 101
```

```
var arr = [0, 1, 2];
console.log(arr.length); // 3

arr.length = 5;
console.log(arr); // [0, 1, 2, undefined, undefined]

arr.length = 2;
console.log(arr); // [0, 1]
console.log(arr[2]); // undefined
```

배열 함수

- 배열 함수
  - 배열 함수는 굉장히 다양한 함수들이 존재하며 아래와 같은 함수들이 있다.

이름	내용
push	배열의 끝에 원하는 값을 추가
pop	배열의 마지막 주소에 있는 값을 제거하여 반환
shift	배열의 첫번째 주소에 있는 값을 제거하여 반환
concat	두개의 배열을 합친다
join	배열값 사이에 원하는 문자를 삽입
reverse	열을 역순으로 재배치
sort	배열을 정렬
slice	배열의 일부분을 반환
splice	배열값을 추가하거나 제거하여 반환

배열 함수

- push

```
var example = new Array("a", "b", "c");
example.push("d");

console.log(example);
```

```
▶ (4) ["a", "b", "c", "d"]
```

- pop

```
var example = new Array("a", "b", "c");
var x = example.pop();

console.log(example);
console.log(x);
```

```
▶ (2) ["a", "b"]
c
```

배열 함수

- shift

```
var example = new Array("a", "b", "c");
var x = example.shift();

console.log(example);
console.log(x);
```

```
▶ (2) ["b", "c"]
a
```

- concat

```
var example = new Array("a","b","c");
var example2 = new Array("d","e","f");

example = example.concat(example2);
console.log(example);
```

```
▶ (6) ["a", "b", "c", "d", "e", "f"]
```

배열 함수

- join

```
var example = new Array("a", "b", "c");
example = example.join("/");

console.log(example);
```

```
a/b/c
```

- reverse

```
var example = new Array("a", "b", "c");
example.reverse();

console.log(example);
```

```
▶ (3) ["c", "b", "a"]
```

배열 함수

- sort

```
var example = new Array(1,4,2,3,5);
example.sort();

console.log(example);
```

```
▶ (5) [1, 2, 3, 4, 5]
```

- slice

```
var example = [1, 2, 3, 4];
var example2 = example.slice(0, -1);
console.log(example);
console.log(example2);

example2 = example.slice(-2);
console.log(example);
console.log(example2);
```

```
▶ (4) [1, 2, 3, 4]
▶ (3) [1, 2, 3]
▶ (4) [1, 2, 3, 4]
▶ (2) [3, 4]
```

배열 함수

- splice

```
var example = ["a", "b", "c", "d"];
var example2 = example.splice(1, 2);

console.log(example);
console.log(example2);
```

```
▶ (2) ["a", "d"]
▶ (2) ["b", "c"]
```

배열 함수

- 배열 함수와 Length
  - 배열 함수는 Length 프로퍼티를 기반으로 동작한다.

```
// arr 배열 생성
var arr = ['zero', 'one', 'two'];

// push() 메서드 호출
arr.push('three');
console.log(arr); // [‘zero’, ‘one’, ‘two’, ‘three’]

// length 값 변경 후, push() 메서드 호출
arr.length = 5;
arr.push('four');
console.log(arr); // [‘zero’, ‘one’, ‘two’, ‘three’', undefined, 'four']
```

```
▶ (4) ["zero", "one", "two", "three"]
▶ (6) ["zero", "one", "two", "three", empty, "four"]
```

객체와 배열의 차이점

- 객체와 배열의 차이점

```
// colorsArray 배열
var colorsArray = ['orange', 'yellow', 'green'];
console.log(colorsArray[0]); // orange
console.log(colorsArray[1]); // yellow
console.log(colorsArray[2]); // green

// colorsObj 객체
var colorsObj = {
    '0': 'orange',
    '1': 'yellow',
    '2': 'green'
};
console.log(colorsObj[0]); // orange
console.log(colorsObj[1]); // yellow
console.log(colorsObj[2]); // green
```

```
// typeof 연산자 비교
console.log(typeof colorsArray); // object
console.log(typeof colorsObj);  // object

// length 프로퍼티
console.log(colorsArray.length); // 3
console.log(colorsObj.length); // undefined

// 배열 표준 메서드
// ['orange', 'yellow', 'green', 'red']
colorsArray.push('red');
// Uncaught TypeError: Object #<Object> has no method 'push'
colorsObj.push('red');
```

객체와 배열의 차이점

- 객체와 배열의 차이점

```
var emptyArray = []; // 배열 리터럴을 통한 빈 배열 생성
var emptyObj = {}; // 객체 리터럴을 통한 빈 객체 생성

console.dir(emptyArray.__proto__); // 배열의 프로토타입(Array.prototype) 출력
console.dir(emptyObj.__proto__); // 객체의 프로토타입(Object.prototype) 출력
```

객체와 배열의 차이점

- 배열에 객체 프로퍼티 삽입
  - 배열에 객체 프로퍼티를 삽입할 수 있다.
  - 객체 프로퍼티를 선언할 경우 이전의 배열을 선언한 변수를 마치 객체 변수 다루듯이 다루면 된다.
  - 단 중요한 점은 객체 프로퍼티가 삽입된다 하여도 배열의 Length에는 전혀 지장이 없다.

객체와 배열의 차이점

- 배열에 객체 프로퍼티 삽입

```
// 배열 생성
var arr = ['zero', 'one', 'two'];
console.log(arr.length); // 3

// 프로퍼티 동적 추가
arr.color = 'blue';
arr.name = 'number_array';
console.log(arr.length); // 3

// 배열 원소 추가
arr[3] = 'red';
console.log(arr.length); // 4

// 배열 객체 출력
console.dir(arr);
```

```
3
3
4
▼Array(4) i
    0: "zero"
    1: "one"
    2: "two"
    3: "red"
    color: "blue"
    name: "number_array"
    length: 4
  ▶ __proto__: Array(0)
```

배열의 순회

- 배열의 순회 방법
  - 배열의 순회 방법에는 여러가지가 있으며 다음과 같이 4가지의 방법이 존재한다.
    - 일반 for 문
    - for ~ in 문
    - for ~ of 문
    - foreach 함수

배열의 순회

- 배열의 순회 방법 - 일반 for 문
  - 일반 for문은 기존 배열의 length 속성을 통해 index를 추출하여 배열의 순회가 가능하다.

```
var items = ['item1', 'item2', 'item3'];

for (let index = 0; index < items.length; index++) {
    console.log(items[index]);
}
```

```
item1
item2
item3
```

배열의 순회

- 배열의 순회 방법 - for ~ in 문
  - 기존 for 문과는 달리 in문은 배열의 처음부터 끝까지 순회할 수 있는 인덱스를 생성한다.
  - 해당 인덱스를 이용한 배열 안의 데이터를 처음부터 끝까지 순회가 가능하다.

```
var items = ['item1', 'item2', 'item3'];

for (const key in items) {
    console.log(`${key} : ${items[key]}`);
}
```

```
0 : item1
1 : item2
2 : item3
```

배열의 순회

- 배열의 순회 방법 - For ~ of
  - For ~ of 문은 반복할 수 있는 객체(iterable Object) 순회할 수 있도록 해주는 반복문이다.
  - For ~ in 문과 반복하는 것은 차이가 없지만 대상과 방법에서 차이가 있다.

```
[구문]
For ( variable of iterableObject ) { 코드 }
```

  - iterable object 를 반복할 때 마다 반복할 수 있는 객체안의 아이템 값이 variable 변수에 설정된다.
  - For ~ of 문을 사용하기 위해서는 객체가 [Symbol.iterator] 속성을 가지고 있어야만 한다.
  - 표현식을 작성할 수 있으며 표현식 평과 결과를 iterable object 로 사용한다.

배열의 순회

- 배열의 순회 방법 - For ~ of 문
  - 인덱스를 사용할 이유가 없고 아이템 값만 받아 사용할 경우 유용하게 쓸 수가 있다.

```
var items = ['item1', 'item2', 'item3'];

for (const item of items) {
    console.log(item);
}
```

```
item1
item2
item3
```

```
for (var value of "ABC"){
  console.log(value);
}
```

```
A
B
C
```

```
for (const value of [10, 20, 30]){
  console.log(value);
}
```

```
10
20
30
```

배열의 순회

- 배열의 순회 방법 - For ~ of 문
  - DOM을 활용한 객체 순회.

```
<body>
    <ul>
        <li>첫 번째</li>
        <li>두 번째</li>
        <li>세 번째</li>
    </ul>
    <script type="text/javascript">
        let nodes = document.querySelectorAll("li");
        for (var node of nodes) {
            console.log(node.textContent);
        };
    </script>
</body>
```

```
첫 번째
두 번째
세 번째
```

배열의 순회

- 배열의 순회 방법 - For ~ of 문
  - for ~ in : 객체의 모든 열거 가능한 속성에 대해 반복
  - for ~ of : [Symbol.iterator] 속성을 가지는 컬렉션 전용

```
let values = [10, 20, 30];
Array.prototype.music = function(){
  return "음악"
};

Object.prototype.sports = function(){
  return "스포츠"
};
console.log("<<<for-in>>>");
for (var key in values) {
  console.log(key, values[key]);
};
console.log("<<<for-of>>>");
for (var value of values) {
  console.log(value);
};
```

```
<<<for-in>>>
0 10
1 20
2 30
music ƒ (){
sports ƒ (){
<<<for-of>>>
10
20
30
```

배열의 순회

- 배열의 순회 방법 - For ~ of 문
  - Object 객체는 이더러블 오브젝트가 아니므로 for~of 문으로 열거할 수 없다.
  - 하지만 개발자 코드로 사전 처리를 하면 for~of 문으로 열거가 가능하다.

```
let sports = {
  soccer: "축구",
  baseball: "야구"
};

let keyList = Object.keys(sports);
for (var key of keyList){
  console.log(key, sports[key]);
};
```

```
soccer 축구
baseball 야구
```

배열의 순회

- 배열의 순회 방법 – foreach(callback)
  - Array에서 제공하는 함수
  - 처음부터 끝까지 순회하고자 할 때 쓰는 함수이며 안쪽 파라미터에는 callback function 이 들어간다.
  - callback function 의 파라미터로 두개를 사용할 수가 있는데 ([해당 데이터 아이템],[인덱스번호]) 두개를 선언하여 사용이 가능하다.

```
var items = ['item1', 'item2', 'item3'];

items.forEach(function(item, index){
    console.log(`${index} : ${item}`);
});
```

```
0 : item1
1 : item2
2 : item3
```

배열과 객체의 차이점

- for문, for ~ in, for ~ of, forEach 문을 사용한 배열과 객체 순회
  - 실제 for문이나 for ~ of 혹은 forEach() 함수를 활용하여 객체를 순회할 수는 없다.
  - 이유는 위에서 나온 for문들의 경우 이터러블(iterable), 즉 순회 가능한 형태의 객체가 아니기 때문이다.
  - 하지만 for ~ in 을 사용할 경우 기존의 for문들하고 달리 객체도 순회가 가능하다.
  - 만약 객체의 아이템들을 순회하거나 혹은 Array 객체 안에 넣은 프로퍼티까지도 순회할 경우 for ~ of를 쓰는 편이 좋다.

배열과 객체의 차이점

- for문, for ~ in, for ~ of, forEach 문을 사용한 배열과 객체 순회

```
// 일반 배열 생성
var arr1 = ['item1', 'item2', 'item3'];

// 일반 객체 생성
var arr2 = {
    0 : 'item1',
    1 : 'item2',
    2 : 'item3'
}

// 배열 생성 후 프로퍼티 추가
var arr3 = ['item1', 'item2', 'item3'];
arr3.color = 'blue';
arr3.name = 'number_array';
```

```
// for문 순회
console.log(`-------------일반 for문------------`);
for (let index = 0; index < arr1.length; index++) console.log(`단순 배열 순회 : ${arr1[index]}`);
for (let index = 0; index < arr2.length; index++) console.log(`단순 객체 순회 : ${arr2[index]}`);
for (let index = 0; index < arr3.length; index++) console.log(`복합 배열 객체 순회 : ${arr3[index]}`);

// for~in 문 조회
console.log(`-------------for in문------------`);
for (const key in arr1) console.log(`단순 배열 순회 ${key}:${arr1[key]}`);
for (const key in arr2) console.log(`단순 객체 순회 ${key}:${arr2[key]}`);
for (const key in arr3) console.log(`복합 배열 객체 순회 ${key}:${arr3[key]}`);

// for~of 문 조회
console.log(`-------------for of문------------`);
for (const item of arr1) console.log(`단순 배열 순회 ${item}`);
// error : arr2 is not iterable
// for (const item of arr2) console.log(`단순 객체 순회 ${item}`);
for (const item of arr3) console.log(`복합 배열 객체 순회 ${item}`);

// foreach 문 조회
console.log(`-------------foreach문------------`);
arr1.forEach(function(item, key){ console.log(`단순 배열 순회 ${key}:${item}`);});
// error : arr2 is not a function
// arr2.forEach(function(item, key){ console.log(`단순 배열 순회 ${key}:${item}`); });
arr3.forEach(function(item, key){ console.log(`복합 배열 배열 순회 ${key}:${item}`);});
```

배열과 객체의 차이점

- for문, for ~ in, for ~ of, forEach 문을 사용한 배열과 객체 순회
  - 출력 결과

```
------------일반 for문------------
단순 배열 순회 : item1
단순 배열 순회 : item2
단순 배열 순회 : item3
복합 배열 객체 순회 : item1
복합 배열 객체 순회 : item2
복합 배열 객체 순회 : item3
------------for in문------------
단순 배열 순회 0:item1
단순 배열 순회 1:item2
단순 배열 순회 2:item3
단순 객체 순회 0:item1
단순 객체 순회 1:item2
단순 객체 순회 2:item3
복합 배열 객체 순회 0:item1
복합 배열 객체 순회 1:item2
복합 배열 객체 순회 2:item3
복합 배열 객체 순회 color:blue
복합 배열 객체 순회 name:number_array
------------for of문------------
단순 배열 순회 item1
단순 배열 순회 item2
단순 배열 순회 item3
복합 배열 객체 순회 item1
복합 배열 객체 순회 item2
복합 배열 객체 순회 item3
------------foreach문------------
단순 배열 순회 0:item1
단순 배열 순회 1:item2
단순 배열 순회 2:item3
복합 배열 배열 순회 0:item1
복합 배열 배열 순회 1:item2
복합 배열 배열 순회 2:item3
```

배열 요소 삭제

- 배열 요소 삭제
  - 배열도 객체이므로, 배열 요소나 프로퍼티를 삭제하는데 delete 연산자를 사용할 수 있다.
  - 단 delete를 사용하여도 length 값은 변하지 않으며 완전히 삭제할 경우에는 splice() 메서드를 사용한다.

```
var arr = ['zero', 'one', 'two', 'three'];
delete arr[2];
console.log(arr); // ["zero", "one", undefined × 1 , "three"]
console.log(arr.length); // 4
```

```
▶ (4) ["zero", "one", empty, "three"]
4
```

유사배열객체(Array Like)

- 유사배열객체(Array-Like)
  - Array는 아니지만 Array 처럼 사용할 수 있는 Object를 Array-Like라고 한다.

```
[구문]
let values = { 0 : “zero” , 1 : “one” , 2 : “two” , length : 3 };
```

  - 배열은 인덱스(index)를 갖고 있으며 작성된 순서대로 읽기, 수정, 삭제가 가능하다.
  - 또한, 엘리먼트의 수를 나타내는 length 라는 프로퍼티가 존재하며 해당 배열의 길이를 나타낸다.
  - { key : value } 형태의 Object 특징을 유지하면서 배열의 특징을 가미한 것이 Array-like이다.

유사배열객체(Array Like)

- 유사배열객체(Array-Like)

```
let values = {0: "zero", 1: "one", 2: "two", length: 3};

for (var key in values){
  console.log(key, ':', values[key]);
};

for (var k = 0; k < values.length; k++){
  console.log(values[k]);
};
```

```
0 : zero
1 : one
2 : two
length : 3
zero
one
two
```

  - Array-like는 순차적으로 작성이 되어야 한다.

```
let values = {10: "ten", zoo: "동물원", 2: "two", length: 3};
for (var k = 0; k < values.length; k++){
  console.log(values[k]);
};
```

```
2 undefined
  two
```

유사배열객체(Array Like)

- 유사배열객체(Array Like)
  - 유사 배열 객체의 가장 큰 특징은 객체임에도 불구하고, apply나 call 메서드를 통해 자바스크립트의 표준 배열 메서드를 사용하는 게 가능하다는 것이다.

```
var arr = ['bar'];
var obj = {
    name : 'foo',
    length : 1
};

arr.push('baz');
console.log(arr);  // ['bar', 'baz']

obj.push('baz');  // Uncaught TypeError: Object #<Object> has no method 'push'
```

유사배열객체(Array Like)

- 유사배열객체(Array Like)

```
var arr = ['bar'];
var obj = { name : 'foo', length : 1};

arr.push('baz');
console.log(arr); // ['bar', 'baz']

Array.prototype.push.apply(obj, ['baz']);
console.log(obj); // { '1': 'baz', name: 'foo', length: 2 }
```

```
▶ (2) ["bar", "baz"]
▶ {1: "baz", name: "foo", length: 2}
```

Spread 연산자

- Spread 연산자
  - 스프레드(Spread) 연산자는 이터러블 오브젝트와 엘리먼트를 하나씩 분리하여 전개할 때 쓰이는 연산자이다.
  - 열이나 문자열과 같이 반복 가능한 문자를 0개 이상의 인수 (함수로 호출할 경우) 또는 요소 (배열 리터럴의 경우)로 확장하여, 0개 이상의 키-값의 쌍으로 객체로 확장시킬 수 있다.
  - 전개한 결과를 변수에 할당하거나 호출하는 함수의 파라미터 값으로 사용할 수 있다.

[구문]
[. . . iterableObject]

Spread 연산자

- Spread 연산자 기본 예제

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

console.log(arr); // [ 1, 2, 3, 4, 5 ]
console.log(...arr); // 1, 2, 3, 4, 5
console.log(1, 2, 3, 4, 5); // 1, 2, 3, 4, 5

const a = [...arr];
console.log(a) // [1, 2, 3, 4, 5];
```

```
const aArr = [1, 2, 3];
const bArr = [4, 5, 6];

// array.concat
console.log(aArr.concat(bArr)); // [ 1, 2, 3, 4, 5, 6 ]

// spread
console.log(...aArr, ...bArr); // 1, 2, 3, 4, 5, 6

// spread to array
console.log([...aArr, ...bArr]); // [ 1, 2, 3, 4, 5, 6 ]
```

Spread 연산자

- Spread 연산자 기본 예제

```
let one = [11, 12];
let two = [21, 22];
let spreadObj = [51, ...one, 52, ...two];

console.log(spreadObj); // [51, 11, 12, 52, 21, 22]
console.log(spreadObj.length); // 6
```

```
const values = [10, 20, 30, 40];
get(...values);

function get(one, two, three){
    console.log( one + two + three); // 60
};
```

```
let spreadObj = [..."music"];
console.log(spreadObj); // ['m', 'u', 's', 'i', 'c']
```

Spread 연산자

- Spread 연산자 기본 예제
  - Spread 연산자를 활용한 객체 assign() 함수 구현

```
var obj1 = { foo: 'bar', x: 42 };
var obj2 = { foo: 'baz', y: 13 };

var clonedObj = { ...obj1 };
console.log(clonedObj);
// Object { foo: "bar", x: 42 }

var mergedObj = { ...obj1, ...obj2 };
console.log(mergedObj);
// Object { foo: "baz", x: 42, y: 13 }
```

```
> {foo: 'bar', x: 42}
> {foo: 'baz', x: 42, y: 13}
```

Default Value

- Default Value
  - 변수, 파라미터, 프로퍼티에 값이 할당되지 않을 때 사전에 정의한 값이 할당되도록 하는 기능
  - 따라서 이 기능을 사용하면 변수, 파라미터, 프로퍼티에 값을 주지 않을 경우 해당 변수, 파라미터, 프로퍼티는 디폴트 값을 갖게 된다.

```
let [one, two, five = 5] = [1, 2];
console.log(one, two, five); // 1, 2, 5

[one, two, five = 5] = [1, 2, 77];
console.log(one, two, five); // 1, 2, 77

let {six, seven = 7} = {six: 6};
console.log(six, seven); // 6, 7
```

```
let [one, two = one + 1, five = two + 3] = [1];
console.log(one, two, five); // 1, 2, 5
```