배열

2020-01-19

이승진

학습목표

배열

배열의 메소드

2차원 배열

목차

[1. 배열 2](#_Toc36576750)

[1) 배열 만들기 2](#_Toc36576751)

[2) 배열 원소의 타입 3](#_Toc36576752)

[3) 빈 배열 만들기 3](#_Toc36576753)

[4) 배열 크기 자동 확대 4](#_Toc36576754)

[5) 배열은 객체 4](#_Toc36576755)

[6) 구조 분해 할당 (destructing assignment) 5](#_Toc36576756)

[2. 배열의 메소드 7](#_Toc36576757)

[1) toString 메소드 7](#_Toc36576758)

[2) splice 메소드 7](#_Toc36576759)

[3) slice 메소드 9](#_Toc36576760)

[4) push, pop 메소드 10](#_Toc36576761)

[5) unshift, shift 메소드 11](#_Toc36576762)

[6) concat 메소드 12](#_Toc36576763)

[7) sort 13](#_Toc36576764)

[8) reverse 14](#_Toc36576765)

[3. 다차원 배열 15](#_Toc36576766)

[1) 2차원 배열 #1 15](#_Toc36576767)

[2) 2차원 배열 #2 17](#_Toc36576768)

[3) 3차원 배열 17](#_Toc36576769)

[4. 요약 18](#_Toc36576770)

[5. 과제 19](#_Toc36576771)

[1) sort.js 19](#_Toc36576772)

[2) even.js 19](#_Toc36576773)

[6. 메모리 구조 20](#_Toc36576774)

[1) 배열 메모리 구조 20](#_Toc36576775)

[2) 2차원 배열 #2 21](#_Toc36576776)

[3) 2차원 배열 #2 21](#_Toc36576777)

[4) 2차원 배열 #3 22](#_Toc36576778)

# 배열

## 배열 만들기

### array1.js

|  |
| --- |
| let a = [1, 2, 3, 4];  for (let i = 0; i < a.length; ++i) {      console.log(a[i]);  } |

출력

|  |
| --- |
| 1  2  3  4 |

### array2.js

|  |
| --- |
| let a = ["one", "two", "three", "four"];  for (let i = 0; i < a.length; ++i) {      console.log(a[i]);  } |

출력

|  |
| --- |
| one  two  three  four |

### length 속성

배열의 length 속성은 배열의 길이를 리턴한다.

## 배열 원소의 타입

javascript 언어는 약타입 언어이므로, 배열 원소의 타입에 제약이 없다.

즉 배열 원소가 모두 같은 타입이지 않아도 된다.

### array3.js

|  |
| --- |
| let a = [3, "hello", true];  for (let i = 0; i < a.length; ++i) {    console.log(typeof a[i]);  } |

출력

|  |
| --- |
| number  string  boolean |

## 빈 배열 만들기

길이 0 인 배열 만들기.

### array4.js

|  |
| --- |
| let a = [];  console.log(a.length); |

출력

|  |
| --- |
| 0 |

let a = [];

지역 변수 a 에 빈 배열이 대입된다.

## 배열 크기 자동 확대

배열의 길이를 초과하는 위치(index)에 값을 대입하면, 배열의 크기가 자동으로 확대된다.

### array5.js

|  |
| --- |
| let a = [3];  console.log("최초 크기 : " + a.length);  a[5] = 456;  console.log("새 크기 : " + a.length);  for (let i = 0; i < a.length; ++i)    console.log("a[%d] : %s", i, a[i]); |

출력

|  |
| --- |
| 최초 크기 : 1  새 크기 : 6  a[0] : 3  a[1] : undefined  a[2] : undefined  a[3] : undefined  a[4] : undefined  a[5] : 456 |

let a = [3];

최초 a 배열의 크기는 1 이고, a[0] 위치에 3 이 들어있다.

a[5] = 456;

이 줄을 실행하는 순간, a 배열의 크기는 6으로 자동 확대된다.

그리고 a[5] 위치에 456 값이 대입된다.

a[1] 부터 a[4] 까지에는 아직 아무 값도 대입되지 않았기 때문에,

이 위치의 값은 undefined 이다.

## 배열은 객체

### array6.js

|  |
| --- |
| let a1 = [1, 2, 3];  let a2 = ["a", "b", "c"];  console.log(typeof a1);  console.log(typeof a2); |

출력

|  |
| --- |
| object  object |

javascript 언어에서 배열은 객체(object) 이다.

배열이 객체이므로, 메소드를 호출할 수 있다.

## 구조 분해 할당 (destructing assignment)

### destructing1.js

|  |
| --- |
| let a = [3, 4, 5];  let [e1, e2, e3] = a;  console.log("%d %d %d", e1, e2, e3);  [e1, e2] = a;  console.log("%d %d", e1, e2); |

출력

|  |
| --- |
| 3 4 5  3 4 |

### destructing assignment

배열의 원소들 각각을 동시에 변수들에 대입한다.

let [e1, e2, e3] = a;

e1, e2, e3 지역 변수가 생성된다.

a[0] 값이 e1 변수에 대입된다.

a[1] 값이 e2 변수에 대입된다.

a[2] 값이 e3 변수에 대입된다.

[e1, e2] = a;

이미 존재하는 e1, e2 변수를 사용한다.

a[0] 값이 e1 변수에 대입된다.

a[1] 값이 e2 변수에 대입된다.

### destructing2.js

|  |
| --- |
| let a = [3, 4, 5, 6, 7, 8];  let [e1, e2, ...e3] = a;  console.log("%d %d", e1, e2);  console.log(e3); |

출력

|  |
| --- |
| 3 4  [ 5, 6, 7, 8 ] |

let [e1, e2, ...e3] = a;

e1, e2, e3 지역 변수가 생성된다.

a[0] 값이 e1 변수에 대입된다.

a[1] 값이 e2 변수에 대입된다.

a[2] 부터 배열 끝까지의 값들이 들어있는 배열이 생성되고, 이 배열이 e3 변수에 대입된다.

# 배열의 메소드

## toString 메소드

### toString.js

|  |
| --- |
| let a1 = [1, 2, 3];  let a2 = ["a", "b", "c"];  console.log(a1.toString());  console.log(a2.toString()); |

출력

|  |
| --- |
| 1,2,3  a,b,c |

배열 객체의 toString 메소드는, 배열의 내부 값을 표현하는 string을 리턴한다.

## splice 메소드

### splice 메소드의 파라미터

배열.splice(삽입할\_위치, 삭제할\_항목\_수, 삽입할\_값\_목록)

### splice1.js

|  |
| --- |
| let a = [0, 1, 2];  a.splice(1, 0, "a");  console.log(a); |

출력

|  |
| --- |
| [ 0, 'a', 1, 2 ] |

a.splice(1, 0, "a");

a 배열의 인덱스 1 위치에서, 0 개의 항목을 삭제하고, "a" 항목을 삽입한다.

### splice2.js

|  |
| --- |
| let a = [0, 1, 2];  a.splice(1, 0, "a", "b");  console.log(a); |

출력

|  |
| --- |
| [ 0, 'a', 'b', 1, 2 ] |

a.splice(1, 0, "a", "b");

a 배열의 인덱스 1 위치에서, 0 개의 항목을 삭제하고, "a", "b" 항목을 삽입한다.

### splice3.js

|  |
| --- |
| let a = [0, 1, 2];  a.splice(1, 1, "a");  console.log(a); |

출력

|  |
| --- |
| [ 0, 'a', 2 ] |

a.splice(1, 1, "a");

a 배열의 인덱스 1 위치에서, 1 개의 항목을 삭제하고, "a" 항목을 삽입한다.

### splice4.js

|  |
| --- |
| let a = [0, 1, 2];  a.splice(1, 2, "a");  console.log(a); |

출력

|  |
| --- |
| [ 0, 'a' ] |

a.splice(1, 2, "a");

a 배열의 인덱스 1 위치에서, 2 개의 항목을 삭제하고, "a" 항목을 삽입한다.

### splice5.js

|  |
| --- |
| let a = [0, 1, 2];  a.splice(1, 1);  console.log(a); |

출력

|  |
| --- |
| [ 0, 2 ] |

a.splice(1, 1);

a 배열의 인덱스 1 위치에서, 1 개의 항목을 삭제하고, 아무것도 삽입하지 않는다.

### 요약: 배열에 항목 한 개 삽입하기

배열.splice(삽입할\_위치, 0, 삽입할\_값);

|  |
| --- |
| let a = [0, 1, 2]  a.splice(1, 0, "a");  console.log(a); |

### 요약: 배열에서 항목 한 개 삭제하기

배열.splice(삭제할\_위치, 1);

|  |
| --- |
| let a = [0, 1, 2]  a.splice(1, 1);  console.log(a); |

## slice 메소드

### slice 메소드의 파라미터

.slice(시작\_위치, 끝\_위치)

slice 메소드는 배열의 부분을 리턴한다.

리턴되는 부분 배열은, 원본 배열에서 시작\_위치에서 끝\_위치 사이의 항목들이다.

(시작위치 <= 위치 < 끝 위치)

원본 배열은 변경되지 않는다.

### slice.js

|  |
| --- |
| let a = [0, 1, 2, 3];  console.log(a.slice(0, 1));  console.log(a.slice(0, 2));  console.log(a.slice(1, 2));  console.log(a.slice(1, 3)); |

출력

|  |
| --- |
| [ 0 ]  [ 0, 1 ]  [ 1 ]  [ 1, 2 ] |

## push, pop 메소드

### push.js

|  |
| --- |
| let a = [];  for (let i = 0; i < 5; ++i)    a.push(i);  console.log(a.toString()); |

출력

|  |
| --- |
| 0,1,2,3,4 |

배열 객체의 push 메소드는, 배열의 끝에 항목을 추가한다.

배열의 크기는 1 증가한다.

push 메소드의 리턴 값은 배열의 새 크기이다.

### pop.js

|  |
| --- |
| let a = [1, 3, 5, 7];  while (a.length > 0) {    let value = a.pop();    console.log("pop value=%d, array=[%s]", value, a.toString());  } |

출력

|  |
| --- |
| pop value=7, array=[1,3,5]  pop value=5, array=[1,3]  pop value=3, array=[1]  pop value=1, array=[] |

배열 객체의 pop 메소드는, 배열의 끝의 항목을 배열에서 제거하여 리턴한다.

배열의 크기는 1 감소한다.

## unshift, shift 메소드

### unshift.js

|  |
| --- |
| let a = [];  for (let i = 0; i < 5; ++i)    a.unshift(i);  console.log(a.toString()); |

출력

|  |
| --- |
| 4,3,2,1,0 |

배열 객체의 unshift 메소드는, 배열의 선두에 항목을 추가한다.

배열의 크기는 1 증가한다.

unshift 메소드의 리턴 값은 배열의 새 크기이다.

### shift.js

|  |
| --- |
| let a = [1, 3, 5, 7];  while (a.length > 0) {    let value = a.shift();    console.log("pop value=%d, array=[%s]", value, a.toString());  } |

출력

|  |
| --- |
| pop value=1, array=[3,5,7]  pop value=3, array=[5,7]  pop value=5, array=[7]  pop value=7, array=[] |

배열 객체의 shift 메소드는, 배열의 선두의 항목을 배열에서 제거하여 리턴한다.

배열의 크기는 1 감소한다.

## concat 메소드

두 배열을 결합한 새 배열을 생성하여 리턴한다.

### concat.js

|  |
| --- |
| let a1 = [1, 2];  let a2 = [3, 4];  let a3 = a1.concat(a2);  console.log(a1);  console.log(a2);  console.log(a3); |

출력

|  |
| --- |
| [ 1, 2 ]  [ 3, 4 ]  [ 1, 2, 3, 4 ] |

let a3 = a1.concat(a2);

a1 배열과 a2 배열을 결합한 새 배열을 생성하여 리턴한다.

a1 배열과 a2 배열은 변경되지 않는다.

## sort

### sort1.js

|  |
| --- |
| let a = [3, 1, 6, 2, 4, 8, 10, 5, 11, 7, 9];  a.sort();  console.log(a); |

출력

|  |
| --- |
| [ 1, 10, 11, 2, 3, 4, 5, 6, 7, 8, 9 ] |

배열의 sort 메소드는 배열의 항목들을 정렬한다.

배열에 들어있는 값이 숫자이지만, 문자열인 것처럼 정렬한다.

그래서 정렬한 후 2 보다 10, 11 이 앞에 위치한다.

### sort2.js

|  |
| --- |
| let a = [3, 1, 6, 2, 4, 8, 10, 5, 11, 7, 9];  a.sort(compareNumber);  console.log(a);  function compareNumber(i, j) {    return i - j;  } |

출력

|  |
| --- |
| [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 ] |

sort 메소드의 파라미터로 비교 함수를 전달하면,

이 비교 함수를 기준으로 정렬된다.

비교 함수는 정렬할 두 값을 비교한다.

첫째 파라미터가 둘째 파라미터 보다

크면(정렬 순서상 뒤이면) 양수를 리턴하고,

작으면 (정렬 순서상 앞이면) 음수를 리턴하고,

두 값이 같으면 0을 리턴해야 한다.

### sort3.js

|  |
| --- |
| let a = [3, 1, 6, 2, 4, 8, 10, 5, 11, 7, 9];  a.sort(compareNumber);  console.log(a);  function compareNumber(i, j) {    return j - i;  } |

출력

|  |
| --- |
| [ 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 ] |

비교함수 내부에서 i, j 순서를 바꿔서, 내림차순 정렬을 구현하였다.

## reverse

### reverse1.js

|  |
| --- |
| let a = [];  for (let i = 0; i < 5; ++i)    a.push(i)  a.reverse()  console.log(a) |

출력

|  |
| --- |
| [ 4, 3, 2, 1, 0 ] |

배열의 reverse 메소드는 배열의 항목들의 순서를 뒤집는다.

# 다차원 배열

## 2차원 배열 #1

### array7a.js

|  |
| --- |
| let a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];  for (let i = 0; i < a.length; ++i)    for (let j = 0; j < a[i].length; ++j)      console.log(a[i][j]); |

출력

|  |
| --- |
| 1  2  3  4  5  6  7  8  9 |

a[0] 값은 [1, 2, 3] 일차원 배열이다.

a[1] 값은 [4, 5, 6] 일차원 배열이다.

a[2] 값은 [7, 8, 9] 일차원 배열이다.

a 값은 3행 3열 2차원 배열이다.

### array7b.js

|  |
| --- |
| let a = [];  a[0] = [1, 2, 3];  a[1] = [4, 5, 6];  a[2] = [7, 8, 9];  for (let i = 0; i < a.length; ++i)    for (let j = 0; j < a[i].length; ++j)      console.log(a[i][j]); |

array7b.js 의 실행 결과와 메모리 구조는 array7a.js 와 동일하다.

let a = [];

최초 a 배열의 크기는 0 이다.

a[0] = [1, 2, 3];

이 줄을 실행하는 순간, a 배열의 크기는 1로 자동 확대된다.

a[0] 값은 [1, 2, 3] 일차원 배열이다.

a 배열은 1행 3열 2차원 배열이다.

a[1] = [4, 5, 6];

이 줄을 실행하는 순간, a 배열의 크기는 2로 자동 확대된다.

a[1] 값은 [4, 5, 6] 일차원 배열이다.

a 배열은 2행 3열 2차원 배열이다.

a[2] = [7, 8, 9];

이 줄을 실행하는 순간, a 배열의 크기는 3으로 자동 확대된다.

a[2] 값은 [7, 8, 9] 일차원 배열이다.

a 배열은 3행 3열 2차원 배열이다.

### array7c.js

|  |
| --- |
| let a1 = [1, 2, 3];  let a2 = [4, 5, 7];  let a3 = [7, 8, 9];  let a = [a1, a2, a3];  for (let i = 0; i < a.length; ++i)    for (let j = 0; j < a[i].length; ++j)      console.log(a[i][j]); |

array7c.js 의 실행 결과와 메모리 구조는 array7a.js 와 동일하다.

a 배열은 3행 3열 2차원 배열이다.

## 2차원 배열 #2

### array8.js

|  |
| --- |
| let a = [[1, 2], [3, 4, 5], [6, 7, 8, 9]];  for (let i = 0; i < a.length; ++i)    for (let j = 0; j < a[i].length; ++j)      console.log(a[i][j]); |

출력

|  |
| --- |
| 1  2  3  4  5  6  7  8  9 |

2차원 배열에서 열의 크기가 모두 같지 않아도 된다.

a 배열은 3행 ?열 2차원 배열이다.

a[1] 값은 [1, 2] 일차원 배열이다.

a[2] 값은 [3, 4, 5] 일차원 배열이다.

a[3] 값은 [6, 7, 8, 9] 일차원 배열이다.

## 3차원 배열

### array9.js

|  |
| --- |
| let a = [[[1, 2], [3, 4]], [[5, 6], [7, 8]], [[9, 10], [11, 12]]];  for (let i = 0; i < a.length; ++i)    for (let j = 0; j < a[i].length; ++j)      for (let k = 0; k < a[i][j].length; ++k)        console.log(a[i][j][k]); |

출력

|  |
| --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 |

# 요약

배열의 length 속성은 배열의 길이를 리턴한다.

javascript 언어는 약타입 언어이므로, 배열 원소의 타입에 제약이 없다.

즉 배열 원소가 모두 같은 타입이지 않아도 된다.

let a = [];

지역 변수 a 에 빈 배열이 대입된다.

배열의 길이를 초과하는 위치(index)에 값을 대입하면, 배열의 크기가 자동으로 확대된다.

배열에서 아직 아무 값도 대입되지 않은 위치의 값은 undefined 이다.

javascript 언어에서 배열은 객체(object) 이다.

2차원 배열에서 열의 크기가 모두 같지 않아도 된다.

배열 객체의 toString 메소드는, 배열의 내부 값을 표현하는 string을 리턴한다.

splice(삽입할\_위치, 삭제할\_항목\_수, 삽입할\_값\_목록)

배열에 항목 한 개 삽입하기

배열.splice(삽입할\_위치, 0, 삽입할\_값);

배열에서 항목 한 개 삭제하기

배열.splice(삭제할\_위치, 1);

배열.slice(시작\_위치, 끝\_위치)

slice 메소드는 배열의 부분을 리턴한다.

리턴되는 부분 배열은, 원본 배열에서 시작\_위치에서 끝\_위치 사이의 항목들이다.

(시작위치 <= 위치 < 끝 위치)

원본 배열은 변경되지 않는다.

배열 객체의 push 메소드는, 배열의 끝에 항목을 추가한다.

배열의 크기는 1 증가한다.

push 메소드의 리턴 값은 배열의 새 크기이다.

배열 객체의 pop 메소드는, 배열의 끝의 항목을 배열에서 제거하여 리턴한다.

배열의 크기는 1 감소한다.

배열 객체의 unshift 메소드는, 배열의 선두에 항목을 추가한다.

배열의 크기는 1 증가한다.

unshift 메소드의 리턴 값은 배열의 새 크기이다.

배열 객체의 shift 메소드는, 배열의 선두의 항목을 배열에서 제거하여 리턴한다.

배열의 크기는 1 감소한다.

2차원 배열에서 열의 크기가 모두 같지 않아도 된다.

# 과제

## sort.js

다음 기능을 구현하시오.

(1) 빈 배열을 만든다.

(2) Math.random() 메소드를 사용하여, 1 이상 100 이하의 정수 난수를 100개 생성해서, 배열에 채운다.

(3) 배열을 오름차순으로 정렬한다.

(4) 배열을 출력한다.

## even.js

다음 기능을 구현하시오.

(1) 빈 배열을 만든다.

(2) Math.random() 메소드를 사용하여, 1 이상 100 이하의 정수 난수를 100개 생성해서, 배열에 채운다.

(3) 배열에 원소들 중에서 짝수(2의 배수)를 찾아서 모두 제거한다.

(4) 배열을 출력한다.

# 메모리 구조

## 배열 메모리 구조

### 숫자 배열

|  |
| --- |
| let a = [1, 2, 3, 4]; |



### 문자열 배열

|  |
| --- |
| let a = ["one", "two", "three", "four"]; |



정확한 메모리 구조는 위 그림이다.

그런데 아래 그림으로 이해해도 된다. 별 차이가 없다.



## 2차원 배열 #2

|  |
| --- |
| let a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]; |



|  |
| --- |
| let a = [];  a[0] = [1, 2, 3];  a[1] = [4, 5, 6];  a[2] = [7, 8, 9]; |

메모리 구조는 위 그림과 동일하다.

## 2차원 배열 #2

|  |
| --- |
| let a1 = [1, 2, 3];  let a2 = [4, 5, 7];  let a3 = [7, 8, 9];  let a = [a1, a2, a3]; |



## 2차원 배열 #3

|  |
| --- |
| let a = [[1, 2], [3, 4, 5], [6, 7, 8, 9]]; |

