

# 10. 문자열과 배열

---

# 문자열에 접근하기

---

# 원시 유형에서 메서드 사용하기

- 자료형 중 단순히 값만 가지고 있을 때 → 원시 유형(primitive type)
- 원시 유형에서도 프로퍼티와 메서드를 사용할 수 있다.

```
str = "hello"  
str.length
```

str은 문자열(string)인데,  
length 프로퍼티는 어디에서 온 것일까?

숫자형과 논리형, 문자열 유형은 별도로 객체가 만들어져 있다.

Number, Boolean, String 객체 ← 래퍼 객체라고 부름

boolean → Boolean 객체

number → Number 객체

string → String 객체

## 오토박싱(autoboxing)

number나 boolean, string 같은 원시 유형에서 프로퍼티나 메서드를 사용하면 일시적으로 원시 유형을 해당 객체로 변환한다.

프로퍼티나 메서드의 사용이 끝나면 해당 객체는 메모리에서 사라진다.

필요할 때에만 임시로 객체로 바꿔 사용하고 사용이 끝나면 다시 원시 유형으로 되돌아오는데, 이것을 오토박싱이라고 한다.

# 문자열의 길이 - length

문자열의 길이를 찾을 때에는 length 프로퍼티 사용

`문자열.length`

```
let str = "Good morning!";  
let greeting = "안녕하세요?"  
str.length // 13  
greeting.length // 6
```

# 특정 위치의 문자에 접근하기 - charAt()

- 방법 1) 대괄호 사용 (ES6 이후)
- 방법 2) charAt() 메서드 사용 `문자열.charAt(위치)`

```
str = "Good morning!";  
str.charAt(3) // "d"  
str[5] // "m"
```

# (예) 문자열에 특정 문자가 몇 개 있나?

사용자가 입력한 문자열에 특정 문자가 몇 개 있는지 확인하는 프로그램

이 페이지 내용:

문자열을 입력하세요.

취소확인

이 페이지 내용:

어떤 문자를 체크하겠습니까?

취소확인

"가나다라가가"에는 "가"가 4개 있습니다.

10WcountChar.html, 10WjsWcountChar.js

```
function counting(str, ch) {  
    let count = 0;  
  
    // 문자열 안의 문자를 하나씩 체크합니다.  
    for (let i = 0; i < str.length ; i++) {  
        if (str[i] === ch)    // if(str.charAt(i) === ch)로 작성 가능.  
            count += 1;  
    }  
    return count;  
}  
  
const string = prompt("문자열을 입력하세요.");  
const letter = prompt("어떤 문자를 체크하겠습니까?");  
  
const result = counting(string, letter);  
document.write(`"${string}"에는 "${letter}"가 <span style="color:red">${result}개 </span> 있습니다.`);
```

# 부분 문자열의 위치 찾기 – indexOf()

두 개 이상의 단어로 구성된 문자열에는 공백으로 구분되는 여러 개의 부분 문자열이 있을 수 있다. indexOf() 메서드는 괄호 안의 문자열이 나타난 위치를 알려 줌. 찾는 문자열이 없으면 -1을 반환.

```
indexOf(문자열)
```

```
indexOf(문자열, 위치)
```

예) str1라는 문자열에서 부분 문자열 위치 찾기

```
str1 = "Good morning, everyone. Beautiful morning."  
str1.indexOf("morning") // 5  
str1.indexOf("evening") // -1
```

```
str1 = "Good morning, everyone. Beautiful morning."  
first = str1.indexOf("morning") // 5  
str1.indexOf("morning", first+1) // 두번째 morning의 위치
```

# 특정 문자(열)로 시작하는지 확인 – startsWith()

- 문자열이 특정 문자나 문자열로 시작하는지 확인
- 문자나 문자열에서 영문자의 대소문자를 구별하므로 주의해야 한다.

*문자열*.startsWith(*문자 또는 문자열*)

```
str2 = "Hello, everyone."  
str2.startsWith("Hello") // true  
str2.startsWith("hello") // false  
str2.startsWith("He")    // true  
str2.startsWith("Hello, ev") // true
```



# 특정 문자(열)로 끝나는지 확인 – endsWith()

- 문자열이 특정 문자나 문자열로 끝나는지 확인
- 문자나 문자열에서 영문자의 대소문자를 구별하므로 주의해야 한다.

*문자열.endsWith(문자 또는 문자열)*

```
str2 = "Hello, everyone."  
str2.endsWith("everyone.") // true  
str2.endsWith("Everyone.") // false  
str2.endsWith("one.") // true  
str2.endsWith("lo, everyone") // false
```

endsWith() 메서드를 사용할 때 문자열과 함께 길이를 지정할 수 있다

*문자열.endsWith(문자 또는 문자열, 길이)*

```
str2 = "Hello, everyone."  
str2.endsWith("one", 15) // true  
str2.endsWith("lo", 5) // true
```

문자열 길이 5 → Hello

# 특정 문자(열)이 있는지 확인 – includes()

문자열에 특정 문자나 문자열이 있는지 확인. 대소문자 구별하므로 주의.

```
문자열.includes(문자 또는 문자열)
```

ES6 이전> indexOf() 메서드 사용

```
// str2 = "Hello, everyone."  
str2.indexOf("every") !== -1  // true
```

ES6 이후> includes() 메서드 사용

```
// str2 = "Hello, everyone."  
str2.includes("every")  // true
```

# 문자열에서 공백 제거하기

문자열에서 공백이란, [Spacebar]를 눌러 입력한 공백, [Tab]을 눌러 입력한 탭, 줄을 바꾸기 위해 사용한 이스케이프 문자(\n, \r) 등을 말한다.

문자열을 연산에 사용하려면 불필요한 공백을 제거하는 것이 좋다

```
문자열.trim()    // 문자열의 앞뒤 공백 제거
```

```
문자열.trimStart() // 문자열의 앞쪽 공백 제거
```

```
문자열.trimEnd()  // 문자열의 뒤쪽 공백 제거
```

```
str3 = " ab cd ef "
```

```
str3.trim()    // 'ab cd ef'
```

```
str3.trimStart() // 'ab cd ef '
```

```
str3.trimEnd()  // ' ab cd ef'
```

# 문자열의 대소문자 바꾸기

영문자 문자열의 경우에는 문자열을 모두 대문자로, 또는 모두 소문자로 바꿀 수 있다.

```
문자열.toUpperCase() // 문자열을 모두 대문자로 변환
```

```
문자열.toLowerCase() // 문자열을 모두 소문자로 변환
```

```
str4 = "Good morning."
```

```
str4.toUpperCase() // 'GOOD MORNING.'
```

```
str4.toLowerCase() // 'good morning.'
```

# 문자열의 부분 문자열 추출하기 - substring()

- 시작 위치부터 **끝 위치의 직전까지** 추출해서 반환한다.
- 끝 위치를 지정하지 않으면 시작 위치부터 문자열 끝까지 추출해서 반환한다.

*문자열.substring(시작 위치)*

*문자열.substring(시작 위치, 끝 위치)*

```
// str4 = "Good morning."
```

```
str4.substring(5)    // 'morning.'
```

# 문자열의 부분 문자열 추출하기 - substring()

str4 = "Good morning." 에서 Good 부분만 추출하려면?

→ 첫번째 글자(인덱스 0)부터 4번째까지 글자(인덱스 3)까지 추출해야 함

→ `str4.substring(0, 4)`

The diagram illustrates the process of extracting a substring from the string "Good morning.". Above the string, two orange arrows point to specific indices: the first arrow, labeled "시작 위치" (Start Position), points to index 0; the second arrow, labeled "끝 위치" (End Position), points to index 4. Below the string, the characters are indexed from 0 to 12. The characters "G", "o", "o", and "d" at indices 0, 1, 2, and 3 respectively are highlighted with an orange background, representing the substring "Good".

0	1	2	3	4	5	6	7	8	9	10	11	12
G	o	o	d		m	o	r	n	i	n	g	.

```
// str4 = "Good morning."  
str4.substring(0, 4)
```

# 문자열의 부분 문자열 추출하기 - slice()

시작 위치부터 **끝 위치의 직전까지** 추출해서 반환한다.

끝 위치를 지정하지 않으면 시작 위치부터 문자열 끝까지 추출해서 반환한다.

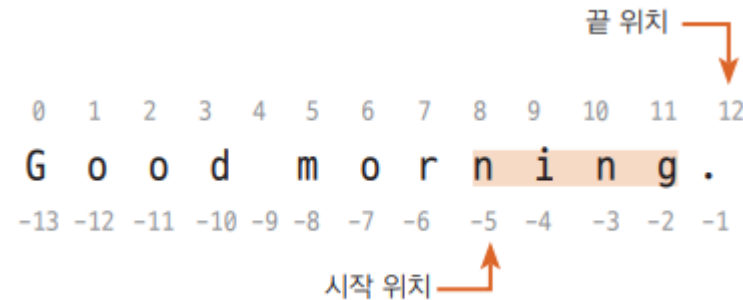
*문자열.slice(시작 위치)*

*문자열.slice(시작 위치, 끝 위치)*

slice() 메서드는 음수를 사용해 위치를 지정할 수 있다.

음수로 지정하면 문자열의 끝에서부터 위치를 찾기 때문에, 문자열을 뒤에서부터 자를 때 편리하다.

```
// str4 = "Good morning."  
str4.slice(-5, 12) // "ning"
```



# 구분자를 사용해 문자 쪼개기 – split()

문자열에서 구분자를 기준으로 문자열을 나눈다.

*문자열.split(구분자)*

```
str5 = "Hello everyone"  
array1 = str5.split(" ") // ["Hello", "everyone"]  
array2 = str.split("")    // ["H","e","l","l","o"," ", "e","v","e","r","y","o","n","e"]
```

↑  
따옴표 사이에 공백이 없음



# [실습] 보안을 위해 메일 주소 일부 감추기

회원제 사이트에서 개인 정보를 표시할 때 보안을 위해, 또는 메일 소유자인지 확인하기 위해 이메일 주소를 전부 보여 주지 않고 일부만 보여줄 경우가 있다.

사용자의 이메일 주소 중 @ 앞의 내용을 세 자리까지만 보여 주는 프로그램을 작성해 보자

**som...@naver.com**

<미리 생각해 보기>

- 메일 주소를 어떤 문자를 기준으로 나누어야 할까
- @기호 앞부분의 문자열을 어떻게 세자리만 남길까

## 문서 구조 살펴보기

10Wemail.html

```
<div id="userInput">  
  <label>  
    <input type="email" id="userEmail" placeholder="이메일 주소를 입력하세요." autofocus>  
  </label>  
  <button>실행</button>  
</div>  
<div id="result"> </div>
```

1) 텍스트 필드와 버튼, 결과값이 표시될 영역을 가져와서 변수에 할당하고 버튼을 클릭했을 때 함수를 실행하도록 한다.

10WjsWemail.js

```
const email = document.querySelector("#userEmail"); // 메일 주소 입력 부분
const button = document.querySelector("button");      // 버튼
const result = document.querySelector("#result");    // 결과 표시 영역
button.addEventListener("click", function() {

});
```

2) 이메일 주소는 @를 기준으로 앞부분은 사용자 이름이고 뒷부분은 도메인 주소이므로 @를 기준으로 문자열을 분리한다

2-1) email.value를 가져와서 split() 메서드 사용

2-2) 사용자 이름 부분은 username에 저장하고, 뒷부분은 domain에 저장

10WjsWemail.js

```
button.addEventListener("click", function() {  
    let username, domain;  
  
    if(email.value !== "") {  
        username = email.value.split("@")[0];    // @ 기준으로 쪼개 앞부분 저장.  
        domain = email.value.split("@")[1];      // @ 기준으로 쪼개 뒷부분 저장.  
    }  
});
```

- 3) username 중에서 세 자리만 필요하므로 첫번째부터 세번째 글자까지만 추출해서 username에 다시 저장
- 4) 수정한 username과 @, domain을 다시 연결

10WjsWemail.js

```
button.addEventListener("click", function() {  
  let username, domain;  
  
  if(email.value !== "") {  
    username = email.value.split("@")[0];  
    username = username.substring(0, 3);  
    domain = email.value.split("@")[1];  
    result.innerText = `${username}...@${domain}`;  
    email.value = "";  
  }  
});
```

## 소스 확장하기

@ 앞의 사용자 이름을 문자 갯수를 고정하지 않고 사용자 이름 길이의 반만 표시하도록 해보자  
앞에서 작성한 이벤트 리스너 부분을 선택해서 주석 처리한다.

[illegible]

10WjsWemail.js

```
button.addEventListener("click", function() {  
    let username, domain, half;  
  
    if(email.value !== "") {  
        username = email.value.split("@")[0];  
        half = username.length / 2;          // username의 길이를 반으로 나눕니다.  
        username = username.substring(0, half);  
        domain = email.value.split("@")[1];  
        result.innerText = `${username}...@${domain}`;  
        email.value = "";  
    }  
});
```

---

**정규 표현식으로 문자열 다루기**

---



# 정규 표현식

- 정규 표현식: 특정 패턴을 사용해 문자열을 표현하는 언어  
(예) 온라인 쇼핑몰에서 물건을 주문할 때 입력한 전화번호가 숫자로만 되어 있는지 체크할 수 있고, 'xxxxxxx-xxxx'와 같은 패턴으로 이루어져 있는지 체크할 수도 있다.
- '패턴=규칙'이라고 생각해도 된다.
- 문자열을 검색하거나 문자열에서 특정 문자를 치환할 때도 복잡한 조건문 없이 정규 표현식을 사용하면 편리하다.

## 정규 표현식 작성하기

기본형 패턴[플래그]

RegExp 객체를 사용하거나 슬래시 (/)를 사용해 표현식으로 작성  
정규 표현식은 '패턴'과 '플래그'로 구성. 플래그는 옵션  
패턴과 플래그 사이에는 공백이 없다.

예) 세 자리 숫자인지 체크하는 정규 표현식

```
let regexp = /\d{3}/  
regexp.test("Hello") // false  
regexp.test("123")
```

# 정규 표현식과 메서드

## RegExp 객체의 메서드

정규 표현식	기능
정규식.test(문자열)	정규 표현식에 일치하는 부분 문자열이 있으면 true를, 없으면 false를 반환합니다.
정규식.exec(문자열)	정규 표현식에 일치하는 부분 문자열이 있으면 결과값을 배열 형태로, 없으면 null을 반환합니다.

## 문자열 메서드 중 정규 표현식과 함께 사용하는 메서드

문자열 메서드	기능
match(정규식)	문자열에서 정규 표현식에 일치하는 부분을 찾습니다.
replace(정규식, 바꿀 문자열)	문자열에서 정규 표현식에 맞는 부분 문자열을 찾아서 새로운 문자열로 바꿉니다.

```
let str = "ES2015 is powerful!"
```

```
str.match(/ES6/) // null
```

# 정규 표현식의 플래그

플래그는 문자열을 검색할 때 사용하는 옵션과 비슷하다고 생각하자

플래그	기능
i	영문자의 대소문자를 구별하지 않고 검색합니다.
g	패턴과 일치하는 것을 모두 찾습니다. g 패턴이 없으면 일치하는 패턴 중 첫 번째 패턴만 반환합니다.
m	문자열의 행이 바뀌어도 검색합니다.

```
let str = "ES2015 is powerful!"  
/es/.test(str)    // false  
/es/i.test(str)   // true
```

test() 메서드 : 정규 표현식 조건에 맞는 문자열이 있는지 체크한다.

# 정규 표현식의 문자 클래스

문자 클래스를 사용하면 문자인지, 숫자인지, 혹은 공백 등을 체크할 수 있다

클래스의 종류	기호	설명
숫자 <sup>digit</sup> 클래스	\d	0 ~ 9 사이의 숫자
	\D	숫자가 아닌 모든 문자
공백 <sup>space</sup> 클래스	\s	공백, 탭(\t), 줄 바꿈(\n) 등
	\S	공백이 아닌 모든 문자
단어 <sup>word</sup> 클래스	\w	단어에 들어가는 문자. 숫자와 밑줄 포함
	\W	단어에 들어가지 않는 모든 문자

```
let str = "ES2015 is powerful!"  
str.match(/ES\d/)
```

/ES\d/ : ES라는 문자 뒤에 오는 하나의 숫자만 찾는다.

```
let str = "ES2015 is powerful!"  
str.match(/ES\d\d\d\d/)
```

/ES\d\d\d\d/ : ES 문자 뒤에 숫자 4개까지

# 문자열의 시작과 끝 체크하기

문자열의 시작과 끝 부분을 체크할 때는 ^ 기호와 \$ 기호 사용

```
let hello = "Hello, everyone."
```

```
/^H/.test(hello) // true. H로 시작하는지 체크
```

```
/^h/.test(hello) // false. h로 시작하는지 체크
```

```
let hello = "Hello, everyone."
```

```
/one.$/.test(hello) // true. one.으로 끝나는지 체크
```

```
/e.$/.test(hello) // true. e.으로 끝나는지 체크
```

```
/one$/.test(hello) // false. one으로 끝나는지 체크
```

[ ] 안에 ^ 기호가 있다면 NOT을 의미한다.

```
"ES2015".match(/[^0-9]/g) // ["E", "S"]
```



숫자가 아닌 것을 체크

# 반복 검색하기

{ } 기호는 반복해서 체크하라고 알려주는 기호.

반복 횟수를 지정하거나 최소 반복 횟수, 최대 반복 횟수를 지정할 수 있다.

표현식	기능
패턴{n}	패턴이 n번 반복되는 것을 찾습니다.
패턴{n,}	패턴이 최소 n번 이상 반복되는 것을 찾습니다.
패턴{m,n}	패턴이 최소 m번 이상, 최대 n번 이하로 반복되는 것을 찾습니다.

```
let str = "Oooops"
```

```
str.match(/o{2}/)    // 'oo'. 2번 반복되는 것
```

```
str.match(/o{2,}/)    // 'ooo' 2번 이상 반복되는 것
```

```
str.match(/o{2,4}/i) // 'Oooo'. 대소문자 구별 없이 o가 2번 이상, 4번 이하로 반복되는 것
```

# 표현식에 사용하는 특수 기호

표현식	기능	사용 예
[ ]	식의 시작과 끝	[a-z]: a부터 z까지
^x	x로 시작하는 문자열. 대괄호([ ]) 안에 ^가 있으면 NOT의 의미가 됩니다.	<ul style="list-style-type: none"><li>• ^[0-9]: 숫자로 시작하는 것</li><li>• [^0-9]: 숫자가 아닌 것</li></ul>
x\$	x로 끝나는 문자열	e\$: e로 끝나는 것
x+	x가 한 번 이상 반복되는 문자열	o+: o, oo처럼 o가 한 번 이상 반복되는 것
x?	x가 0번 또는 1번 있는 문자열	x?: y, xy처럼 x가 없거나 한 번 나타나는 것
x*	x가 0번 이상 반복되는 문자열	x*: y, xy, xxy처럼 x가 없거나 여러 번 반복되는 것
.	문자 하나	[x.z] : xyz나 xAz처럼 x와 z 사이에 문자가 하나 있는 것

# 자주 사용하는 정규 표현식

숫자만 가능

```
/^[0-9]+$/
```

양의 정수

```
/^[1-9]\d*$
```

음의 정수

```
/^\-[1-9]\d*$
```

영문자만 가능

```
/^[a-zA-Z]+$/
```

숫자와 영문자만 가능

```
/^[a-zA-Z0-9]+$/
```

한글만 가능

```
/^[가-힣]+$/
```

한글과 영문자만 가능

```
/^[가-힣a-zA-Z]+$/
```

길이가 5~10개

```
/^{5,10}$
```

메일 주소 체크

```
/^[a-z0-9_+.-]+@[a-z0-9-]+\.[a-z0-9]{2,4}$/
```

전화번호 체크(123-456-7890 또는 123-4567-8901)

```
/(\d{3}).*(\d{3}).*(\d{4})/
```

jpg, gif 또는 png 확장자를 가진 그림 파일

```
/([^\s]+(?:\.(jpg|gif|png))\.\2)/
```

1부터 n 사이의 번호(1과 n 포함)

```
/^[1-9]{1}$|^[1-4]{1}[0-9]{1}$|^n$/
```

암호 체크 - 최소 영문 소문자 하나, 대문자 하나, 숫자 하나가 포함되어 있는 문자열(8글자 이상)

```
/(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}/
```



---

# 문자열과 배열 변환하기

---

# 문자열을 배열로 변환하기

## 왜 변환할까?

- 문자열의 메서드는 문자(열)을 체크하거나 일정 크기만큼 부분 문자열을 추출하는 것 뿐.
- 문자열 안의 문자를 수정하는 메서드는 없지만 배열에는 아주 많은 메서드가 있다.
- 문자열을 다룰 때에는 배열로 변환한 후 배열 메서드를 사용하고, 결과 배열을 다시 문자열로 변환한다.

## split() 메서드나 전개 연산자 사용

```
str5 = "Hello, everyone"
array2 = str5.split("") // ["H","e","l","l","o"," ", "e","v","e","r","y","o","n","e"]
array3 = [...str5]
```

## Array.from() 사용

Array.from(*문자열*)

```
array4 = Array.from(str5)
```

# 문자열 배열을 문자열로 변환하기

## join() 메서드 사용

*배열.join(구분자)*

구분자로 공백없이 ""를 사용하면 배열 요소를 연결해서 문자열을 만든다.

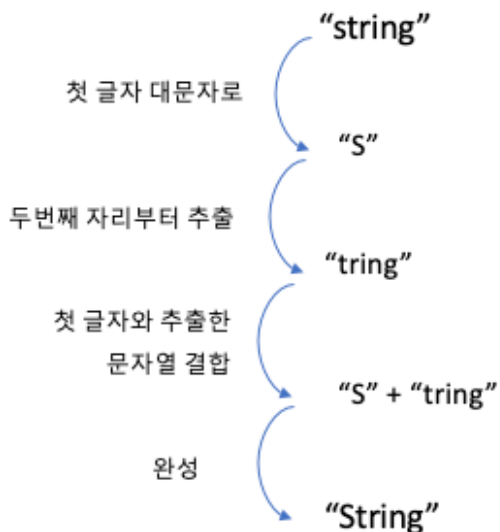
```
str6 = array4.join("")    // "Hello everyone"
```

# [실습] 영문자열의 첫 글자를 대문자로

## <미리 생각해 보기>

- 어떻게 첫 번째 글자와 나머지 문자열을 분리할까?
- 어떻게 글자를 대문자로 바꿀까?
- 분리했던 글자를 어떻게 원래 문자열에 연결할까?

## 방법1)



10Wcapitalize.html, 10WjsWcapitalize.js

```
const string = prompt("영문 소문자로 된 문자열을 입력하세요.");

const firstCh = string[0].toUpperCase();
const remainStr = string.slice(1);
const result = firstCh + remainStr;
document.write(result);
```

## 방법2)

방법1)에서 입력했던 소스를 주석처리하거나 삭제한 후 입력

```
const string = prompt("영문 소문자로 된 문자열을 입력하세요.");  
const result = [string[0].toUpperCase(), ...string.slice(1)].join("");  
document.write(result);
```

2) 두번째부터 끝까지 추출

```
const result = [string[0].toUpperCase(), ...string.slice(1)].join("");
```

1) 첫번째 글자를 대문자로

3) 추출한 문자열 펼쳐 놓음  
[] 안에 표시되므로 배열

4) 1) ~ 3) 의 결과를 연결  
→ 문자열

---

**똑똑하게 배열 사용하기**

---

# 새로운 배열 만들기

## 1) 빈 배열을 만들고 값 할당하기

```
let season = []  
season[0] = "spring"  
season[1] = "summer"  
season // ["spring", "summer"]
```

## 2) 리터럴 표기법으로 만들기

```
let pets = ["dog", "cat", "parrot"]  
pets // ["dog", "cat", "parrot"]
```

## 3) Array 객체의 인스턴스 만들기

```
let fruits = new Array("사과", "복숭아", "포도")  
fruits // ["사과", "복숭아", "포도"]
```

# 배열 값 수정하기 및 추가하기

배열은 인덱스를 사용해서 원하는 위치의 값을 변경할 수 있다.

이미 값이 있는 위치에 값을 할당하면 기존 값은 지워진다.

```
let pets = ["dog", "cat", "parrot"]
pets[1] = "hamster"
pets    // ["dog", "hamster", "parrot"]
```

중간에 인덱스를 건너뛰고 값을 할당할 수 있다.

```
let fruits = new Array('사과', '복숭아', '포도')
fruits[4] = "배"
fruits    // ["사과", "복숭아", "포도", 비어 있음, "배"]
fruits[3] // undefined
```

↑  
empty라고 표시될 수도 있음

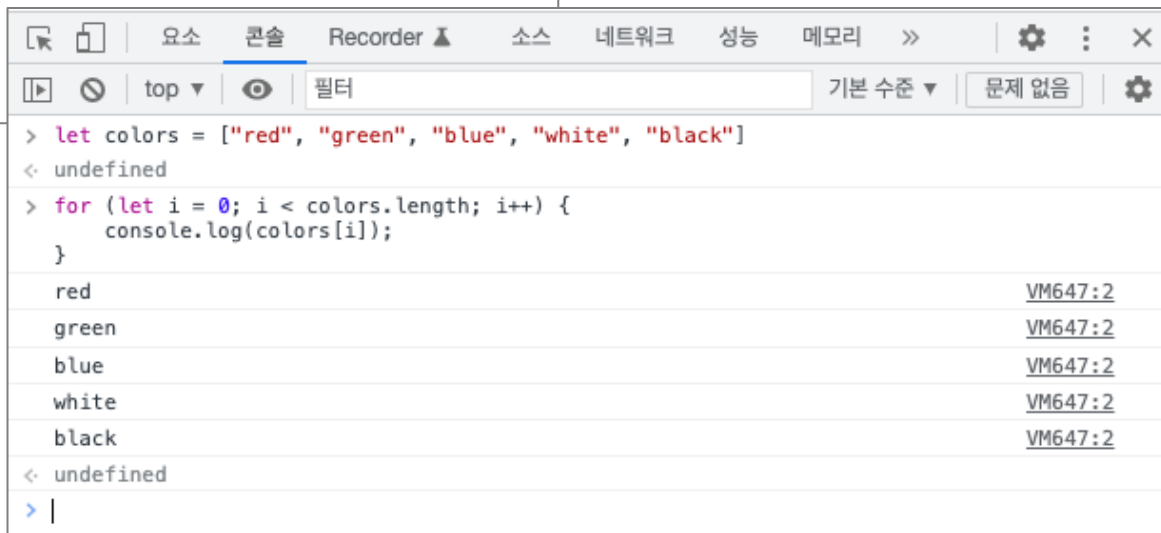


# 배열 요소 순회하기

배열은 여러 개의 값을 가지고 있기 때문에 반복문 사용 가능

## 1) 일반적인 for문

```
let colors= ["red", "green", "blue", "white", "black"]  
  
for (let i = 0; i < colors.length; i++) {  
    console.log(colors[i]);  
}
```



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the execution of a JavaScript script. The first line is `let colors = ["red", "green", "blue", "white", "black"]`, which returns `undefined`. The second line is a `for` loop that iterates over the `colors` array and logs each element. The console shows the following output:

<code>red</code>	<code>VM647:2</code>
<code>green</code>	<code>VM647:2</code>
<code>blue</code>	<code>VM647:2</code>
<code>white</code>	<code>VM647:2</code>
<code>black</code>	<code>VM647:2</code>
<code>undefined</code>	

# 배열 요소 순회하기

## 2) 인수가 1개인 forEach()문

forEach는 배열의 순회를 위해 만들어진 구문

```
배열.forEach(값)
```

예) animals 배열에 있는 각 요소의 값을 표시하려면 배열의 각 요소를 나타내는 animal 변수를 넘겨준다. (보통 배열 이름은 복수, 요소 이름은 단수로 사용)

```
let animals = ["lion", "bear", "bird"]  
animals.forEach(animal => {  
  console.log(animal)  
});
```

배열명                      배열 안의 요소

↓                                      ↓

```
animals.forEach(animal => {  
  console.log(animal)  
});
```

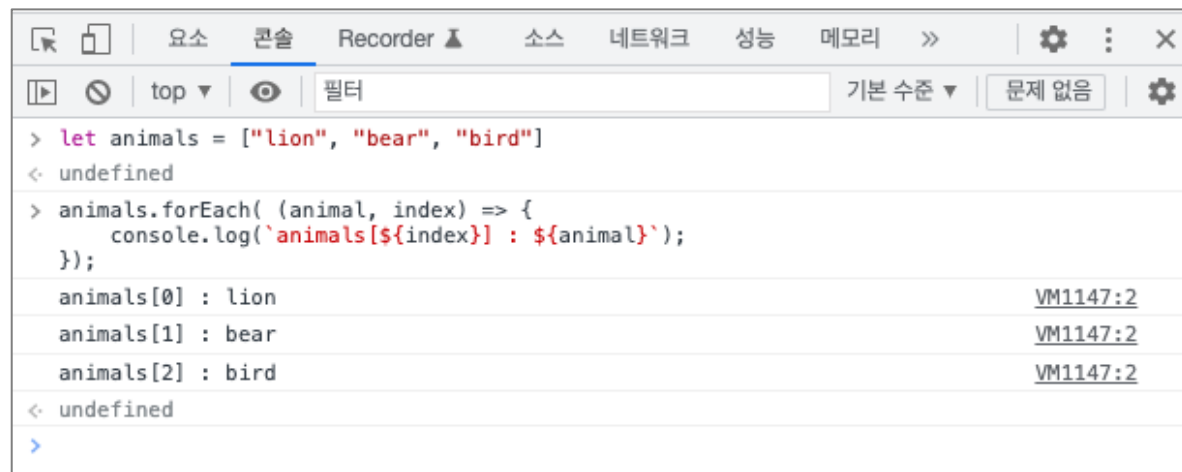
# 배열 요소 순회하기

## 3) 인수가 2개인 forEach()문

요소의 값 뿐만 아니라 인덱스도 필요할 경우 사용

*배열.forEach(값, 인덱스)*

```
animals.forEach((animal, index) => {  
    console.log(`animals[${index}] : ${animal}`);  
});
```



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following code and its output:

```
> let animals = ["lion", "bear", "bird"]  
< undefined  
> animals.forEach( (animal, index) => {  
    console.log(`animals[${index}] : ${animal}`);  
});  
animals[0] : lion VM1147:2  
animals[1] : bear VM1147:2  
animals[2] : bird VM1147:2  
< undefined  
>
```

# 배열 요소 순회하기

## 4) 인수가 3개인 forEach()문

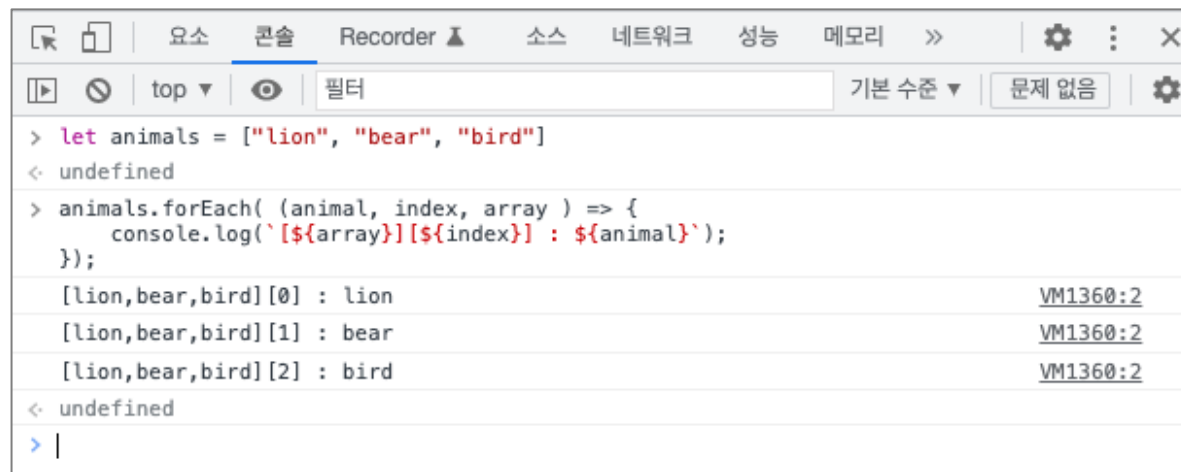
요소의 값과 인덱스, 배열 자신도 필요할 경우 사용

*배열.forEach(값, 인덱스, 배열)*

### for 문과 forEach 문의 가장 큰 차이

for문은 중간에 break문을 사용해 멈출 수 있지만, forEach는 끝까지 다 순회해야 끝난다.

```
animals.forEach((animal, index, array) => {  
    console.log(`${array}[${index}] : ${animal}`)  
});
```



The screenshot shows a web browser's developer console with the 'Console' tab selected. The code being executed is as follows:

```
> let animals = ["lion", "bear", "bird"]  
< undefined  
> animals.forEach( (animal, index, array) => {  
    console.log(`${array}[${index}] : ${animal}`);  
});
```

The console output shows three log statements, each on a new line:

- [lion,bear,bird][0] : lion (VM1360:2)
- [lion,bear,bird][1] : bear (VM1360:2)
- [lion,bear,bird][2] : bird (VM1360:2)

Below the log statements, the console shows the return value of the forEach method, which is undefined.

```
< undefined  
> |
```

# 배열 합치기

기존 배열에 또 다른 배열이나 값을 합쳐서 새로운 배열을 만들 수 있다.

## 1) concat()

```
배열.concat(배열 또는 값, 배열 또는 값, ...)
```

```
let vegetable = ["양상추", "토마토", "피클"]  
let meat = ["불고기"]  
  
let meatBurger = vegetable.concat(meat, "빵")  
meatBurger      // ["양상추", "토마토", "피클", "불고기", "빵"]  
let meatBurger2 = meat.concat("빵", vegetable)  
meatBurger2     // ["불고기", "빵", "양상추", "토마토", "피클"]
```

# 배열 합치기

## 2) 전개 연산자

(ES6 이후에 많이 사용하는 방법)

```
let vegetable = ["양상추", "토마토", "피클"]
```

```
let cheese = ["모짜렐라", "슈레드"]
```

```
let cheeseBurger = ["빵", ...vegetable, ...cheese]
```

```
cheeseBurger // ["빵", "양상추", "토마토", "피클", "모짜렐라", "슈레드"]
```

# 배열 요소 정렬하기

## 역순으로 배치하기 – reverse()

배열 요소의 순서를 거꾸로 바꾸는 메서드로, 값의 크기와는 상관이 없다.

```
배열.reverse()
```

```
let numbers = [6, 9, 3, 21, 15]  
numbers.reverse()           // [15, 21, 3, 9, 6]
```

# 배열 요소 정렬하기

## 크기에 따라 정렬하기 – sort()

sort() 메서드를 사용해서 숫자를 비교해야 한다면 sort() 안에 따로 함수를 정의해야 합니다

*배열.sort(정렬 함수)*

```
let values = [5, 20, 3, 11, 4, 15]

values.sort(function (a, b) {
  if (a > b) return 1;
  if (a < b) return -1;
  if (a === 0) return 0;
});
```

숫자 정렬 함수를 다음과 같이 간단하게 표기할 수 있다.

```
let values = [5, 20, 3, 11, 4, 15]

values.sort(function (a, b) {
  return a - b;
});
```



# 배열 끝에 값, 추가 삭제하기

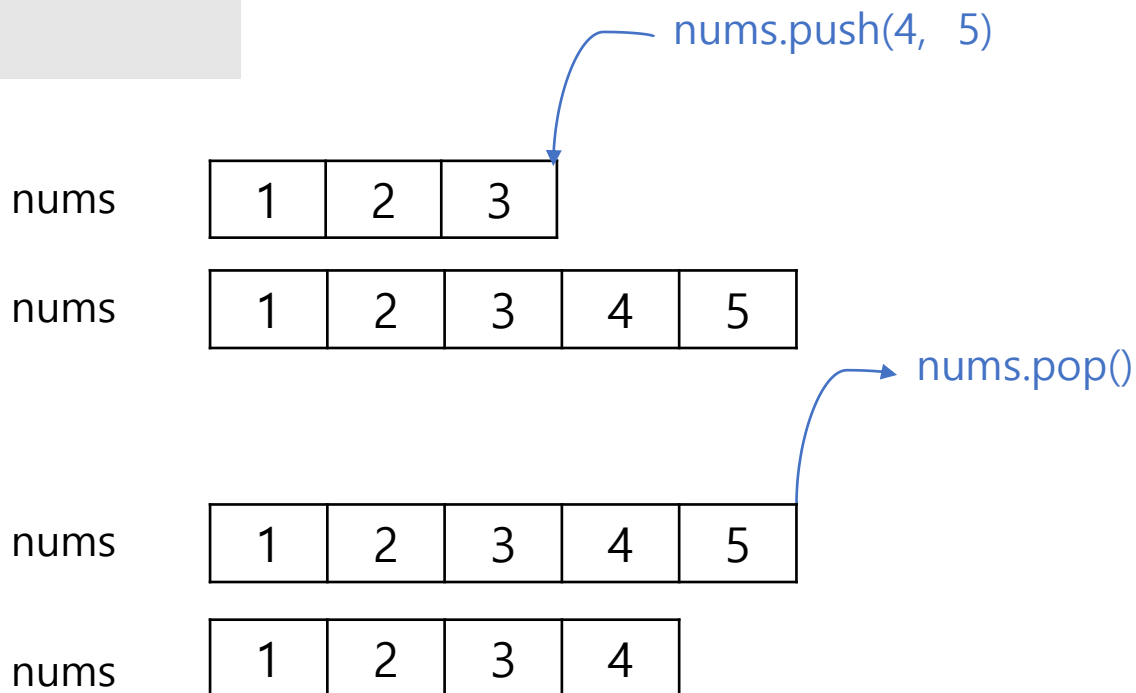
- `push()` : 배열의 맨 앞 부분에 값 추가
- `pop()` : 배열의 맨 끝 값 제거.

`배열.push(값)` // 맨 끝에 값 추가. 배열 개수 반환

`배열.pop()` // 마지막 값 제거.

```
let nums = [1, 2, 3]
nums.push(4, 5)
nums
```

```
nums.pop()
```



# 배열 앞에 값, 추가 삭제하기

unshift() : 배열의 맨 앞에 값 추가

shift() : 배열 맨 앞에 있는 값 제거

```
배열.unshift(값)    // 맨 앞에 값 추가
```

```
배열.shift()        // 맨 앞의 요소 제거
```

```
let fruits = ["apple", "pear", "banana"]
```

```
fruits.shift()    // "apple"
```

```
fruits            // ["pear", "banana"]
```

```
fruits.unshift("cherry")    // 3
```

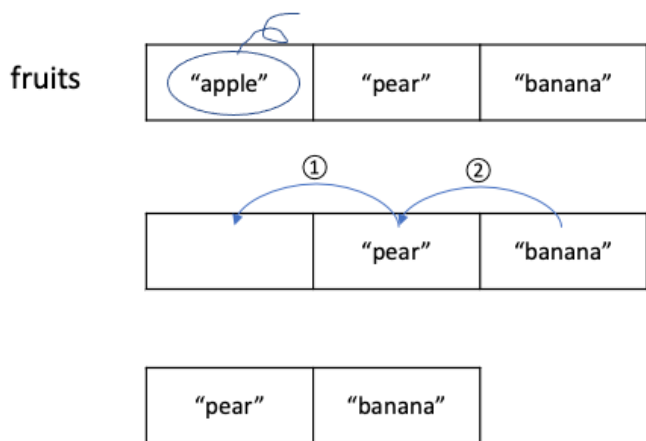
```
fruits                // ["cherry", "pear", "banana"]
```

# 배열 앞에 값, 추가 삭제하기

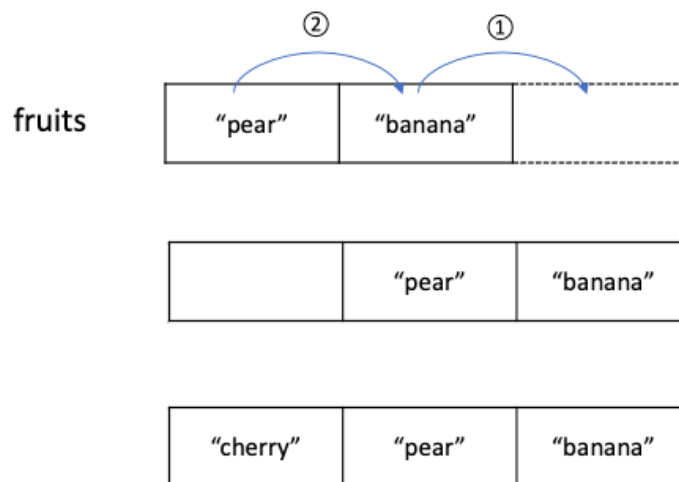
shift() 메서드와 unshift() 메서드는 배열에서 맨 앞의 요소를 변경하기 때문에 요소를 추가하거나 제거하는 작업 외에도 인덱스를 변경해야 한다.

→ 배열의 요소가 많거나 요소의 내용이 복잡할수록 shift(), unshift() 메서드의 실행 시간이 좀 더 길어진다.

**fruits.shift()**



**fruits.unshift("cherry")**



# 원하는 위치에 값, 추가 삭제하기

## 1) splice() 함수

```
배열.splice(위치)           // '위치'부터 끝까지 요소를 삭제합니다.
```

예) subjects라는 배열에서 세 번째 요소부터 끝까지 모두 제거하려면

```
let subjects = ["html", "css", "javascript", "react", "typescript"]  
subjects.splice(2)           // ["javascript", "react", "typescript"]  
subjects                     // ["html", "css"]. 원래 배열이 변경됨
```

# 원하는 위치에 값, 추가 삭제하기

## 1) splice() 함수

```
배열.splice(위치, 숫자) // '위치'에서 '숫자' 개수만큼 요소 삭제
```

예) week 배열에서 인덱스 1번부터 5번까지 추출해서 weekday라는 새로운 배열을 만들 수 있다.  
물론 원래의 week 배열에는 두 개의 요소만 남는다.

```
let week = ["sun", "mon", "tue", "wed", "thu", "fri", "sat"]
let weekday = week.splice(1, 5)
weekday // ["mon", "tue", "wed", "thu", "fri"]
week // ["sun", "sat"]
```

# 원하는 위치에 값, 추가 삭제하기

## 1) splice() 함수

`배열.splice(위치, 숫자, 값)` // '위치'에서 '숫자' 개수만큼 요소를 삭제한 후 '값' 추가

예) 네 개의 요소가 있는 fruits 배열에서 세 번째 자리에 coffee를 추가하려면

```
let brunch = ["egg", "milk", "apple", "banana"]  
  
brunch.splice(2, 0, "coffee", "bread")    // [ ]. 삭제 개수 0  
  
brunch    // ["egg", "milk", "coffee", "bread", "apple", "banana"]
```

# 원하는 위치에 값, 추가 삭제하기

## 2) slice() 함수

```
배열.slice(위치)           // '위치'부터 끝까지 요소를 추출한다.
```

slice() 함수 사용 후에 원래 배열은 변하지 않는다

예) colors 배열에서 blue부터 끝까지 추출해서 새로운 배열을 만들 수 있다

```
let colors= ["red", "green", "blue", "white", "black"]  
  
let colors2 = colors.slice(2)  
  
colors2           // ["blue", "white", "black"]  
  
colors            // ["red", "green", "blue", "white", "black"]
```

# 원하는 위치에 값, 추가 삭제하기

## 2) slice() 함수

*배열*.slice(*위치 1*, *위치 2*) // 위치 1부터 위치 2 직전까지 추출한다.

```
let colors3 = colors.slice(1, 4)    // 위치 1부터 위치 3까지 추출
```

```
colors3          // ["green", "blue", "white"]
```

```
colors           // ["red", "green", "blue", "white", "black"]
```