

# 17. 웹 API 활용하기

---

# API 알아보기

---

# API란

## Application Programming Interface(애플리케이션 프로그래밍을 위한 인터페이스)

'애플리케이션에서 프로그램을 작성할 때 한 프로그램과 다른 프로그램 사이에 쉽게 정보를 주고받을 수 있게 도와주는 인터페이스

예) 캔버스 API로 웹 화면에 쉽게 그래픽을 그릴 수 있다.

예) 기상청 제공 API로 날씨 관련 정보를 제공하는 앱을 만들 수 있다.

## 공개 API

- API 중에서 누구나 사용할 수 있게 만든 것을 공개 'API', 또는 '오픈 API'라고 부른다.
- 공개 API를 사용하면 포털 사이트의 지도 기능이나 인증 기능을 자신의 사이트로 가져와서 넣을 수도 있고, 정부에서 제공하는 각종 자료와 기능을 가져와서 사용할 수도 있다.

# 웹 API

HTML에서 기본으로 제공하는 API.

자바스크립트만 알고 있으면 웹 API를 사용해서 누구나 애플리케이션을 만들 수 있다.

처음에 발표했던 API 외에도 새로운 API가 계속 추가되고 있다

Table of contents	Web API
명세 인터페이스 같이 보기	웹 코드를 작성한다면 많은 API를 사용할 수 있습니다. 아래 목록은 웹 앱이나 웹 사이트를 만들 때 사용할 수 있는 모든 인터페이스(객체와 유형)입니다.  Web API는 보통 JavaScript와 함께 사용하지만, 항상 그렇지는 않습니다.  <b>명세</b>  사용 가능한 API의 전체 목록입니다.
B	<ul style="list-style-type: none"><li>• <a href="#">Background Fetch API (en-US)</a></li><li>• <a href="#">Background Tasks</a></li><li>• <a href="#">Barcode Detection API (en-US)</a></li><li>• <a href="#">Battery API</a> 🗑️</li><li>• <a href="#">Beacon (en-US)</a></li><li>• <a href="#">Bluetooth API (en-US)</a></li><li>• <a href="#">Broadcast Channel API (en-US)</a></li></ul>
C	<ul style="list-style-type: none"><li>• <a href="#">CSS Counter Styles (en-US)</a></li><li>• <a href="#">CSS Font Loading API (en-US)</a></li><li>• <a href="#">CSS Painting API (en-US)</a></li><li>• <a href="#">CSS Typed Object Model API (en-US)</a></li><li>• <a href="#">CSSOM</a></li><li>• <a href="#">Canvas API</a></li><li>• <a href="#">Channel Messaging API</a></li><li>• <a href="#">Clipboard API</a></li></ul>
	<ul style="list-style-type: none"><li>• <a href="#">Network Information API</a></li><li>• <a href="#">Page Visibility API</a></li><li>• <a href="#">Payment Request API (en-US)</a></li><li>• <a href="#">Performance API (en-US)</a></li><li>• <a href="#">Performance Timeline API (en-US)</a></li><li>• <a href="#">Periodic Background Sync (en-US)</a></li><li>• <a href="#">Permissions API (en-US)</a></li><li>• <a href="#">Picture-in-Picture API (en-US)</a></li><li>• <a href="#">Pointer Events (en-US)</a></li><li>• <a href="#">Pointer Lock API (en-US)</a></li><li>• <a href="#">Presentation API (en-US)</a></li><li>• <a href="#">Proximity Events</a> 🗑️ 🗑️</li><li>• <a href="#">Push API</a></li></ul>
	R
	<ul style="list-style-type: none"><li>• <a href="#">Resize Observer API (en-US)</a></li></ul>

[developer.mozilla.org/ko/docs/Web/API](https://developer.mozilla.org/ko/docs/Web/API)

# 웹 API

API 목록 중에서 궁금한 API를 클릭하면 API에 대한 설명과 함께 어떤 프로퍼티와 메서드가 있는지, 메서드는 어떻게 사용하는지 예제를 보면서 학습할 수 있다.

Table of contents

개념과 사용법  
인터페이스  
연관 배열  
예제  
명세  
브라우저 호환성  
같이 보기

Related Topics

Geolocation API

▼ Guides

Using the Geolocation API

▼ Interfaces

Geolocation  
GeolocationCoordinates  
GeolocationPosition  
GeolocationPositionError

▼ Properties

Navigator.geolocation

Geolocation API

Secure context: This feature is available only in secure contexts (en-US) (HTTPS), in some or all supporting browsers.

Geolocation API는 사용자가 원할 경우 웹 애플리케이션에 위치 정보를 제공할 수 있는 API입니다. 개인정보 보호를 위해, 브라우저는 위치 정보를 제공하기 전에 사용자에게 위치 정보 권한에 대한 확인을 받습니다.

Geolocation 객체를 사용하려면 WebExtension은 매니페스트에 "geolocation" 권한을 추가해야 합니다. 사용자의 운영 체제는 WebExtension이 처음으로 위치 정보를 요청하는 순간 사용자에게 정보 제공 여부를 물어봅니다.

개념과 사용법

사용자의 현재 위치를 지도에 표시하거나 위치 기반 개인화 정보를 제공하는 등, 웹 앱에서 위치 정보를 가져와야 하는 경우가 종종 있습니다.

Geolocation API는 `navigator.geolocation` 을 통해 접근합니다. 이 때, 사용자의 브라우저는 위치 정보 접근 권한을 요청하게 되고, 사용자가 허가할 경우 현재 장치에서 사용 가능한 최선의 방법(GPS, WIFI, ...)을 통해 위치를 알아냅니다.

위의 과정이 끝난 후, 코드에서는 몇 가지 다른 방법으로 위치 정보를 가져올 수 있습니다.

- `Geolocation.getCurrentPosition()`: 장치의 현재 위치를 가져옵니다.
- `Geolocation.watchPosition()`: 장치의 위치가 바뀔 때마다, 자동으로 새로운 위치를 사용해 호출할 처리기 함수를 등록합니다.

두 메서드 모두 최대 세 개의 매개변수를 받습니다.

- 필수로 지정하는 성공 콜백: 위치 정보를 성공적으로 가져온 경우, 위치 데이터를 담은 `GeolocationPosition` 객체를 유일한 매개변수로 하여 콜백을 호출합니다.
- 선택적으로 지정하는 실패 콜백: 위치 정보를 가져오지 못한 경우, 실패 원인을 담은 `GeolocationPositionError (en-US)` 객체를 유일한 매개변수로 하여 콜백을 호출합니다.

예) 지오로케이션 API

---

# 웹 스토리지 API

---

# 쿠키 vs 웹 스토리지

## 쿠키

사용자가 웹 사이트에 접속해서 웹 사이트를 서핑하는 동안 사용자 컴퓨터에 저장되는 텍스트 파일  
사용자가 사이트에 접속하면 서버에서 쿠키 정보를 활용해서 좀 더 간편하게 사이트를 사용할 수 있게 한다.  
쿠키 정보는 사용자 컴퓨터의 하드디스크에 텍스트 파일 형태로 최대 300개까지 저장된다.  
(각 도메인당 50개까지 저장할 수 있고, 한 파일의 최대 크기는 4,096byte(약 4kbyte))

## 쿠키의 단점

쿠키에는 웹 사이트뿐만 아니라 접속했던 개인의 정보가 저장되기 때문에 개인의 사생활을 침해할 수 있다.  
사이트 간에 교차 스크립트 같은 기법을 통해 쿠키를 악용할 수도 있고, 보안 문제가 발생할 수도 있다.  
같은 사이트에서 2개 이상의 탭을 열면 둘 이상의 트랜잭션을 추적하기 어렵다.  
파일 크기가 작기 때문에 복잡한 데이터는 저장할 수 없다.

➔ 웹 스토리지 등장

# 쿠키 vs 웹 스토리지

## 웹 스토리지

스토리지도 웹 사이트와 관련된 정보를 저장한다.

쿠키와 다른 점은 사용자가 일부러 스토리지 정보를 서버로 전송하지 않는 이상 서버에서 사용자 PC로 들어와 스토리지 정보를 읽어가지 못한다는 점.

웹 스토리지는 도메인당 2~10MB(보통 5MB)의 데이터를 저장할 수 있다.

**세션 스토리지** : 웹 브라우저 창(또는 탭)을 여는 순간부터 닫을 때까지를 하나의 세션이라고 한다. 세션 스토리지는 세션 동안만 데이터를 기억하고 있다가 세션이 끝나면(웹 브라우저 창이나 탭이 닫히면) 데이터를 모두 지운다.


**로컬 스토리지** : 로컬 스토리지는 세션이 끝나도 계속해서 데이터를 보관하는 스토리지. 로컬 스토리지를 이용하면 웹 브라우저 창을 닫았다가 다시 열어도 저장된 데이터를 확인할 수 있다.



NAVER

naver.com

네이버를 시작페이지로 > | 즐겨찾기 > | 네이버 > | 해피빈 >

 **NAVER**

요소

콘솔

Recorder

Performance insights

애플리케이션

5

애플리케이션

매니페스트

Service Workers

저장용량

저장용량

로컬 스토리지

https://www.naver.com

https://siape.veta.naver.com

세션 스토리지

IndexedDB

웹 SQL

필터

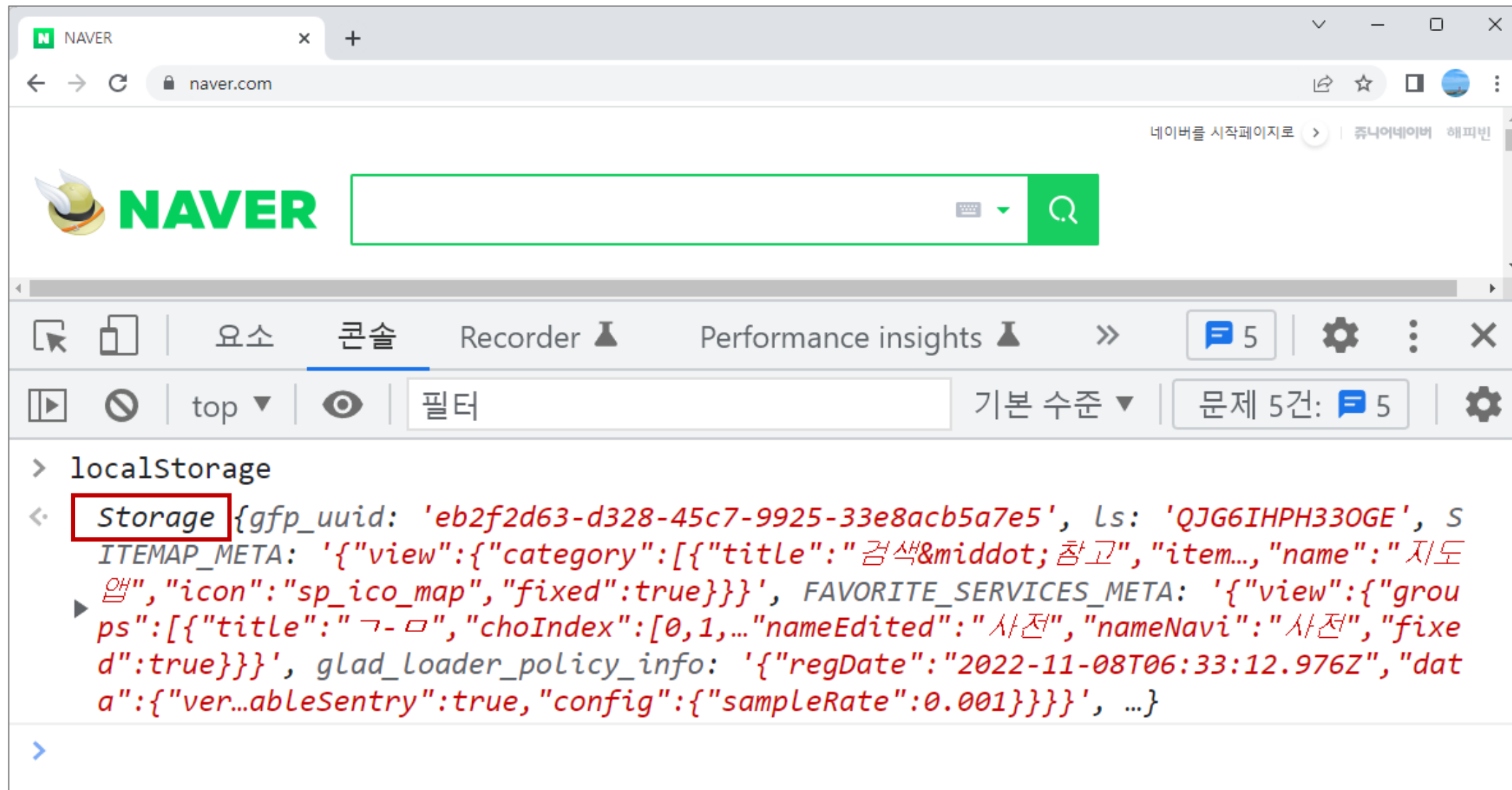
키	값
gfp_uuid	eb2f2d63-d328-45c7-9925-33e8acb5a7e5
ls	QJG6IHPH33OGE
SITEMAP_META	{"view":{"category":[{"title":"검색&middot;참고","it...
FAVORITE_SERVICES_...	{"view":{"groups":[{"title":"ㄱ-ㄴ","choIndex":[0,1,2,...
glad_loader_policy_info	{"regDate":"2022-11-08T06:33:12.976Z","data":{"v...

1

eb2f2d63-d328-45c7-9925-33e8acb5a7e5

## 콘솔 창에서 확인하기

localStorage



The screenshot shows a web browser window with the Naver homepage. The Chrome DevTools console is open, displaying the contents of the `localStorage` object. The `Storage` object is highlighted with a red box. The console output shows the following structure:

```
> localStorage
< Storage {gfp_uuid: 'eb2f2d63-d328-45c7-9925-33e8acb5a7e5', ls: 'QJG6IHPH330GE', SITEMAP_META: '{"view":{"category":[{"title":"검색&middot;참고", "item...", "name":"지도 앱", "icon":"sp_ico_map", "fixed":true}}]', FAVORITE_SERVICES_META: '{"view":{"groups":[{"title":"ㄱ-ㄴ", "choIndex":[0,1,..."nameEdited":"사전", "nameNavi":"사전", "fixed":true}}]', glad_loader_policy_info: '{"regDate":"2022-11-08T06:33:12.976Z", "data":{"ver...ableSentry":true, "config":{"sampleRate":0.001}}}}', ...}
```

# Storage 객체의 프로퍼티와 메서드

- Storage 객체는 sessionStorage 객체와 localStorage 객체를 합쳐서 부르는 용어
- sessionStorage 객체와 localStorage 객체에서 사용하는 프로퍼티와 메서드는 같다.
- 스토리지의 자료는 JSON 문자열을 사용한다.

## length 프로퍼티

스토리지에 몇 개의 키/값 쌍이 있는지 확인.

## setItem() 메서드

기본형 `setItem(키, 값)`

주어진 키에 값을 저장한다. 이미 존재하는 키라면 값을 수정한다.

## getItem() 메서드

기본형 `getItem(키)`

지정한 키에 있는 값을 가져온다. 주어진 키에 해당하는 항목이 없으면 null을 반환한다.

# Storage 객체의 프로퍼티와 메서드

## key() 메서드

기본형    key()  
          key(위치)

스토리지에 있는 키를 반환한다. '위치'를 지정해서 키를 가져올 수도 있다.

## removeItem() 메서드

기본형    removeItem(키)

스토리지에서 지정한 키를 삭제한다. 주어진 키에 해당하는 항목이 없으면 아무 것도 하지 않는다.

## clear() 메서드

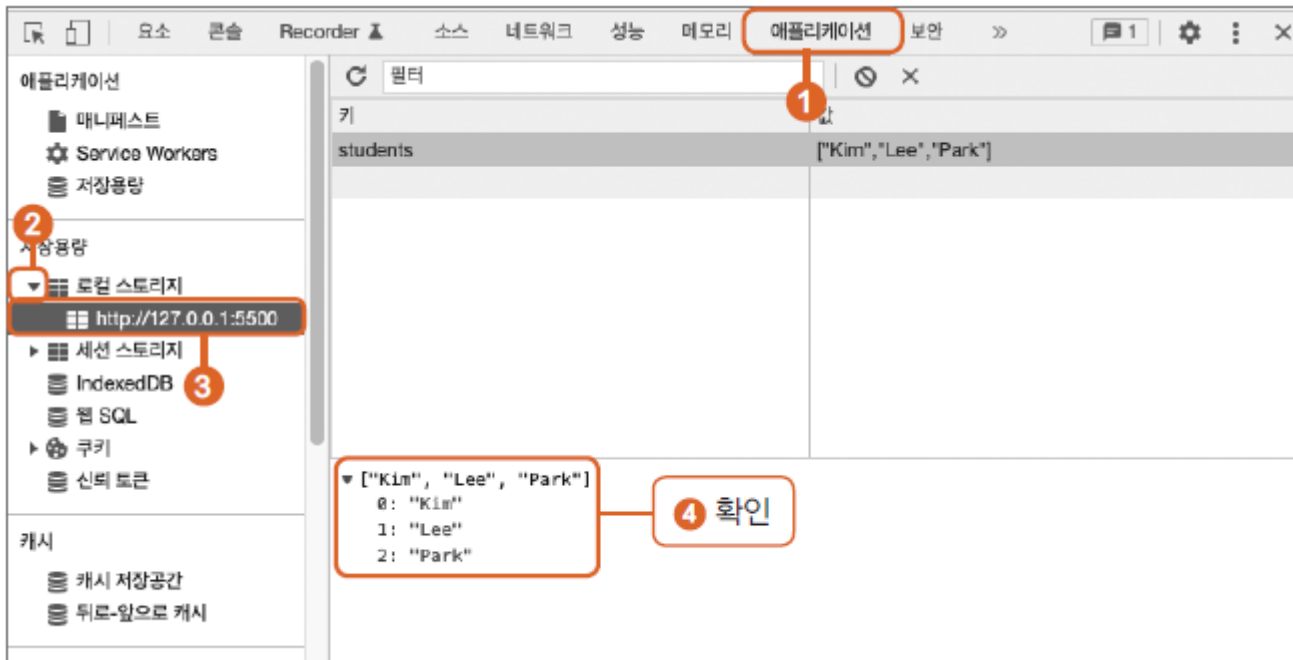
기본형    clear()

스토리지에 있는 모든 키/값 쌍을 삭제한다. 스토리지에 아무 항목도 없으면 아무 것도 하지 않는다.

# [실습] 웹 스토리지 다루기

## 로컬 스토리지에 저장하기

```
let students = ["Kim", "Lee", "Park"];  
console.log(`현재 students : ${students}`);  
// 스토리지에 students 키로 배열을 저장합니다.  
localStorage.setItem("students", JSON.stringify(students));
```



json을 다루므로 VS Code에서 라이브 서버로 열어 확인할 것!

# [실습] 웹 스토리지 다루기

## 로컬 스토리지에 새로운 값 추가하기

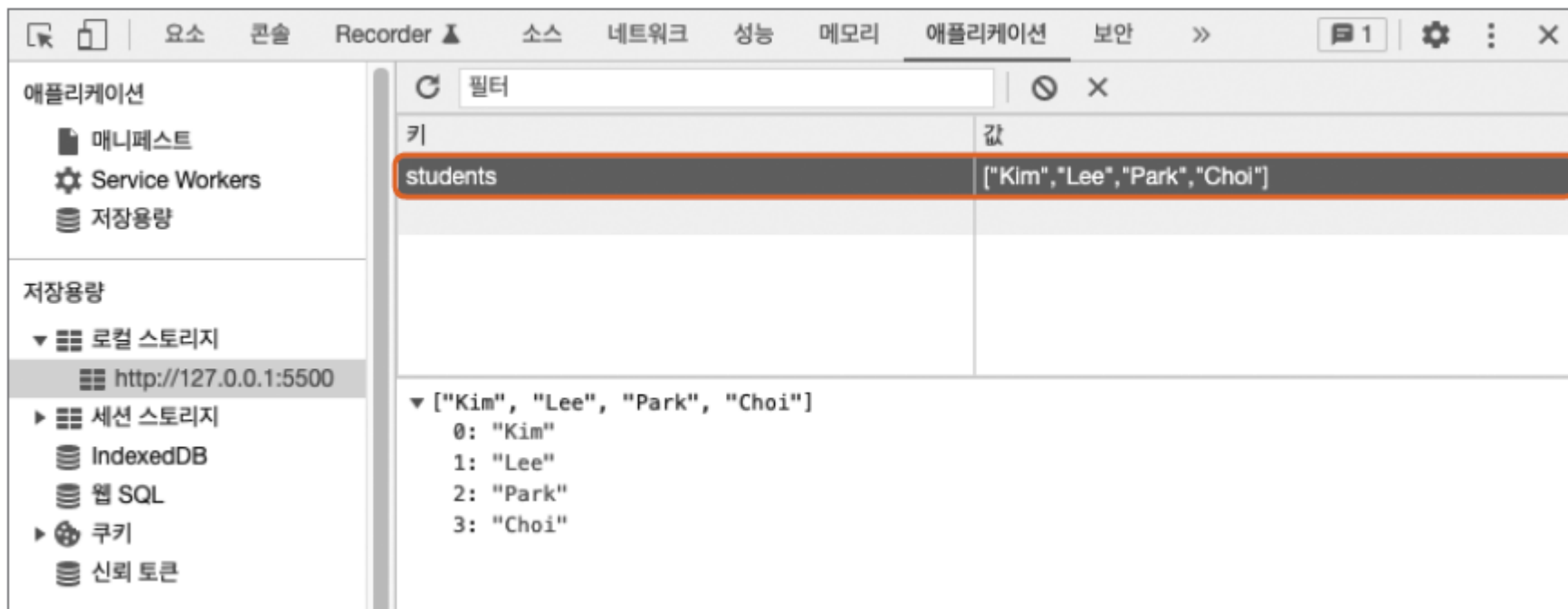
로컬 스토리지에 있는 값을 가져와서 변수에 할당한 후,  
변수에 새로운 값을 추가하고 다시 로컬 스토리지에 저장한다

```
// 로컬 스토리지에 저장합니다.
let students = ["Kim", "Lee", "Park"];
localStorage.setItem("students", JSON.stringify(students)); // 스토리지에 students 키로 배열 저장

// 로컬 스토리지에서 가져온 후 추가하고 저장합니다.
let localData;
if(localStorage.getItem("students") === null) { // 스토리지에 students 키가 있는지 확인.
  localData = [];
} else {
  localData = JSON.parse(localStorage.getItem("students")); // 스토리지의 값을 localData로 저장
}
localData.push("Choi"); // localData에 Choi 추가
localStorage.setItem("students", JSON.stringify(localData)); // 스토리지에 localData 저장
console.log(`추가 후 students : ${localData}`); // 스토리지에 저장된 값을 표시
```

# [실습] 웹 스토리지 다루기

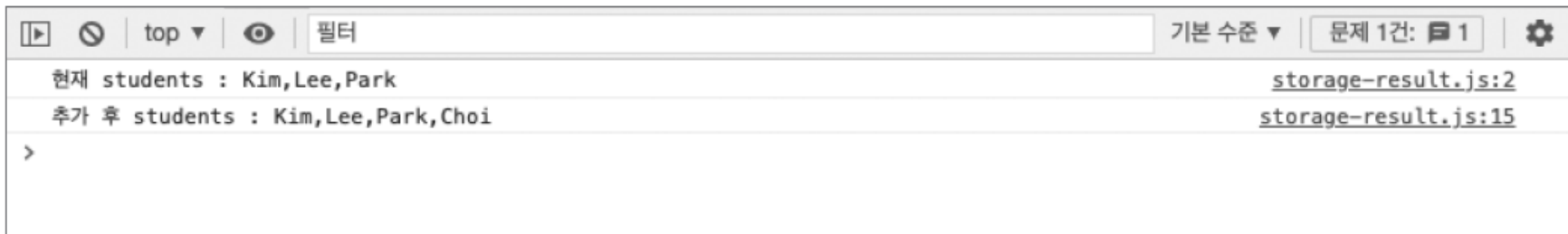
로컬 스토리지에 새로운 값 추가하기



The image shows the Chrome DevTools Application tab. The left sidebar is expanded to 'Local Storage' for the URL 'http://127.0.0.1:5500'. The main panel displays a table of storage items. One item, 'students', is highlighted with an orange border. Below the table, the JSON representation of the 'students' value is shown: an array containing 'Kim', 'Lee', 'Park', and 'Choi'.

키	값
students	["Kim", "Lee", "Park", "Choi"]

▼ ["Kim", "Lee", "Park", "Choi"]  
0: "Kim"  
1: "Lee"  
2: "Park"  
3: "Choi"



The image shows the Chrome DevTools Console. It displays two log messages. The first message shows the current state of 'students' as 'Kim, Lee, Park'. The second message shows the state after adding 'Choi', resulting in 'Kim, Lee, Park, Choi'. The source of the second log is 'storage-result.js:15'.

```
현재 students : Kim, Lee, Park storage-result.js:2  
추가 후 students : Kim, Lee, Park, Choi storage-result.js:15  
>
```

# [실습] 웹 스토리지 다루기

## 로컬 스토리지에서 특정 값 삭제하기

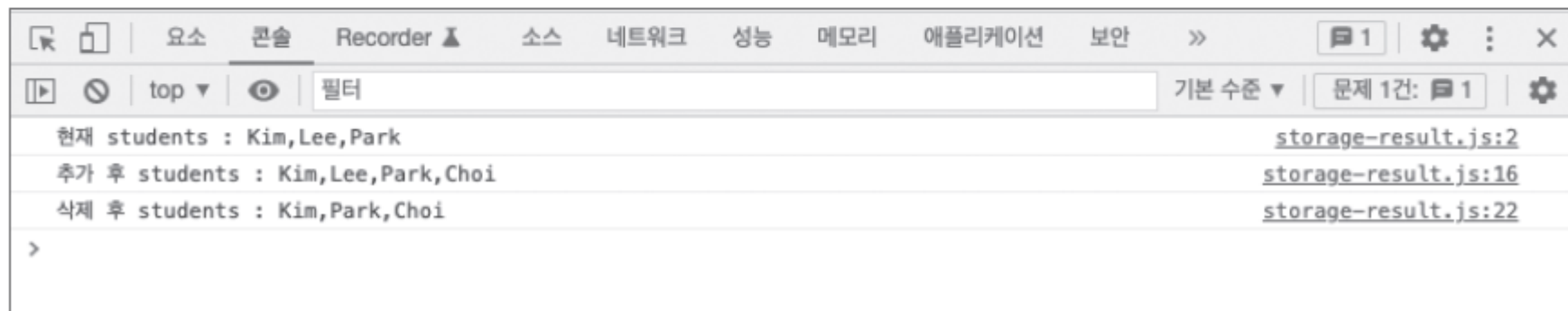
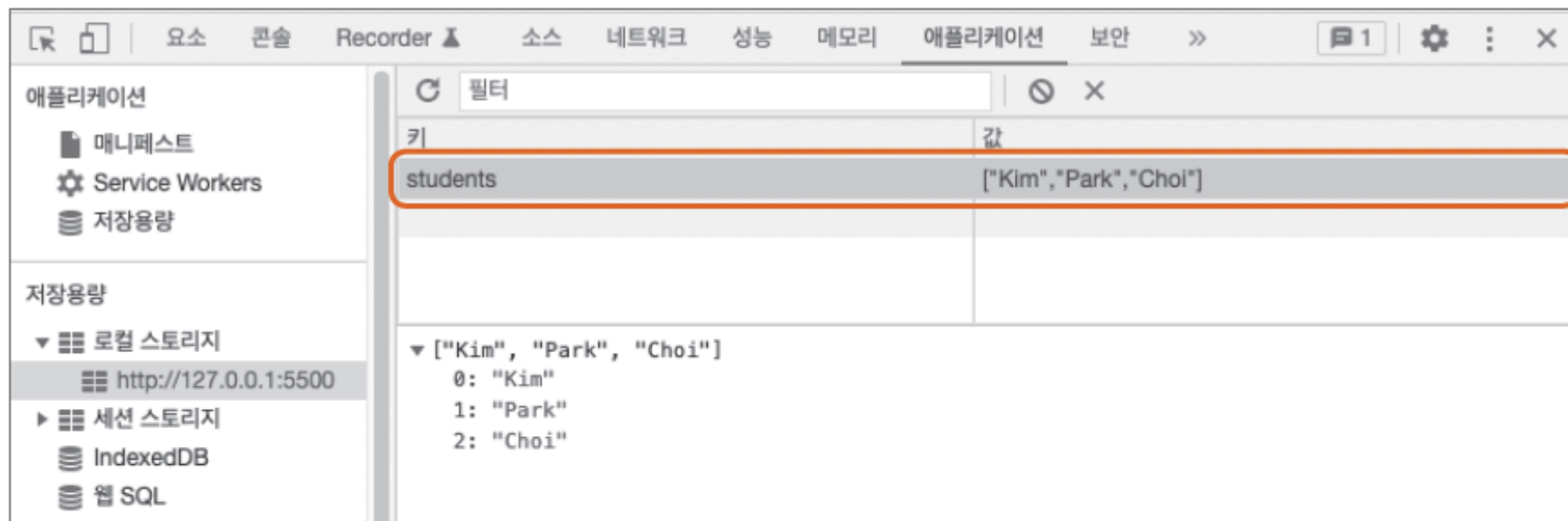
현재 스토리지에는 students 값이 배열 형태로 저장되어 있는데,  
배열에서 특정 값의 위치를 가져올 때는 indexOf() 메서드를 사용한다.  
splice() 메서드를 사용해서 인덱스의 위치부터 1개의 값을 삭제한 후 다시 로컬 스토리지에 저장한다.

```
.....  
// 로컬 스토리지에서 특정 값 삭제하기  
const indexOfValue = localData.indexOf("Lee"); // 인덱스 탐색  
localData.splice(indexOfValue, 1); // 인덱스에 해당하는 값부터 1개 삭제  
localStorage.setItem("students", JSON.stringify(localData));  
console.log(`삭제 후 students : ${localData}`);
```



# [실습] 웹 스토리지 다루기

로컬 스토리지에서 특정 값 삭제하기



# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

결과 화면을 어떻게 표시할지 미리 마크업하고 스타일까지 지정해 둔다

```
<header> ..... </header>
<form> ..... </form>
<div id="container">
  <ul id="todo-list">
    // 이 부분은 결과 화면을 미리 예상해 보는 것이므로 확인 후 삭제한다.
    <div class="todo">
      <li class="todo-content">내용 </li>
      <button class="complete-button">완료</button>
      <button class="delete-button">삭제</button>
    </div>
  </ul>
```

html 파일

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

## css 파일

```
/* todo 항목 */
.todo {
  margin:0.5rem;
  font-size:1.2rem;
  display:flex;
  justify-content: space-between;
  align-items: center;
}
.todo .todo-content {
  flex:1;
  padding:0.5rem 1rem;
  border-bottom: 1px dotted #ccc;
}
```

```
.todo button {
  padding: 0.5rem;
  margin-right:0.2rem;
  font-size:0.8rem;
  cursor: pointer;
}
.completed {
  text-decoration: line-through;
  color: #d8d8d8;
  opacity: 0.5;
}
```

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

브라우저에서 원하는 결과가 나오는지 확인한 후, 마크업했던 부분은 삭제하기

## 오늘의 할 일

추가

내용

완료 삭제

```
<header> ..... </header>
```

```
<form> ..... </form>
```

```
<div id="container">
```

```
  <ul id="todo-list">
```

```
    // 이 부분은 결과 화면을 미리 예상해 보는 것이므로 확인 후 삭제한다.
```

```
    <div class="todo">
```

```
      <li class="todo-content">내용.</li>
```

```
      <button class="complete-button">완료</button>
```

```
      <button class="delete-button">삭제</button>
```

```
    </div>
```

```
</ul>
```

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

웹 요소 가져오기

js 파일

```
// 웹 요소 가져오기
const todoInput = document.querySelector('#todo-input'); // 사용자 입력
const addButton = document.querySelector('#add-button'); // [추가] 버튼
const todoList = document.querySelector('#todo-list'); // 할 일 목록

// 이벤트 처리
addButton.addEventListener('click', addTodo);
```

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

[추가] 버튼을 클릭했을 때 실행할 addTodo 함수

js 파일

```
function addTodo(e) {  
  e.preventDefault(); // 기본 동작 취소  
  
  // 내용 추가  
  const newDiv = document.createElement('div');  
  newDiv.classList.add('todo');  
  const newTodo = document.createElement('li');  
  newTodo.innerText = todoInput.value;  
  newTodo.classList.add('todo-content');  
  newDiv.appendChild(newTodo);
```

```
// 내용의 오른쪽에 버튼 추가  
const completeButton = document.createElement('button');  
completeButton.innerText = '완료';  
completeButton.classList.add('complete-button');  
newDiv.appendChild(completeButton);  
const deleteButton = document.createElement('button');  
deleteButton.innerText = '삭제';  
deleteButton.classList.add('delete-button');  
newDiv.appendChild(deleteButton);  
todoList.appendChild(newDiv);  
todoInput.value = ""; // 입력 창 초기화  
}
```

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

웹 브라우저에서 함수가 제대로 동작하는지 확인

## 오늘의 할 일

자바스크립트 공부하기

친구와 점심

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

로컬 스토리지에 내용 저장하기

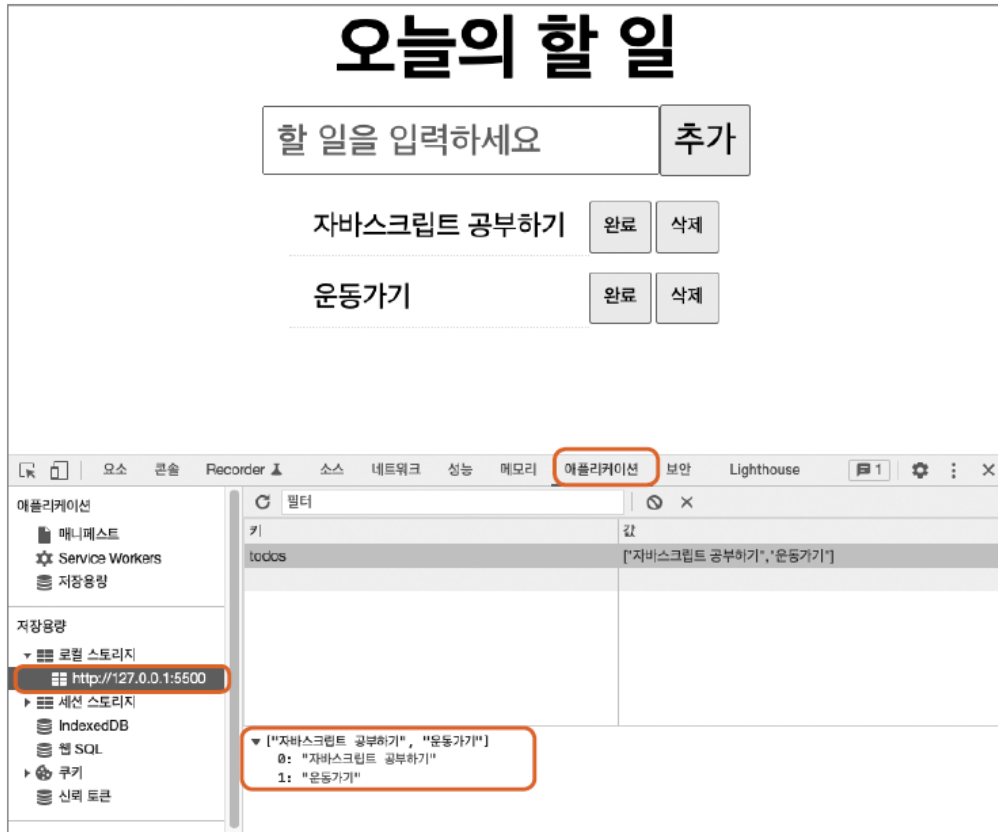
```
function addTodo(e) {  
  .....  
  newTodo.classList.add('todo-content');  
  newDiv.appendChild(newTodo);  
  
  saveToLocal(todoInput.value); // 스토리지에 저장  
  
  .....  
}
```

```
function saveToLocal(todo) {  
  let todos;  
  if (localStorage.getItem('todos') === null) {  
    todos = [];  
  } else {  
    todos = JSON.parse(localStorage.getItem('todos'));  
  }  
  todos.push(todo);  
  localStorage.setItem('todos', JSON.stringify(todos));  
}
```



# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

웹 브라우저에서 확인하기



# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

로컬 스토리지 내용 가져와서 화면에 표시하기

- 로컬 스토리지에 있는 내용은 웹 내용이 모두 로드된 후에 가져와야 한다

```
document.addEventListener("DOMContentLoaded", getLocal);  
addButton.addEventListener('click', addTodo);
```

```
function getLocal() {  
  let todos;  
  if (localStorage.getItem('todos') === null) {  
    todos = [];  
  } else {  
    todos = JSON.parse(localStorage.getItem('todos')); // 스토리지에서 todos 값 가져오기  
  }  
  todos.forEach(function(todo) { // todos 요소마다 반복  
    const newDiv = document.createElement( ' div ' );  
    newDiv.classList.add( ' todo ' );  
    const newTodo = document.createElement( ' li ' );  
    newTodo.innerText = todo; // 로컬 스토리지 값 표시  
    newTodo.classList.add('todo-content');  
    newDiv.appendChild(newTodo);  
    const completeButton = document.createElement('button');  
    completeButton.innerText = '완료';  
    completeButton.classList.add('complete-button');  
    newDiv.appendChild(completeButton);  
    const deleteButton = document.createElement('button');  
    deleteButton.innerText = '삭제';  
    deleteButton.classList.add('delete-button');  
    newDiv.appendChild(deleteButton);  
    todoList.appendChild(newDiv);  
    todoInput.value = "";  
  });  
}
```

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

버튼 클릭했을 때의 동작 정의하기

```
document.addEventListener("DOMContentLoaded", getLocal);  
addButton.addEventListener('click', addTodo);  
todoList.addEventListener('click', manageTodo);
```

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

버튼 클릭했을 때의 동작 정의하기

```
function manageTodo(e) {  
  const whichButton = e.target.classList[0]; // 클릭한 부분의 class명 가져오기  
  if(whichButton === 'complete-button') { // [완료] 버튼이면  
    const todo = e.target.parentElement;  
    todo.children[0].classList.toggle('completed'); // 내용 부분에 .completed 클래스 토글  
  } else if(whichButton === 'delete-button') { // [삭제] 버튼이면  
    const todo = e.target.parentElement; // [삭제] 버튼의 부모 요소를 todo에 할당  
    removeLocal(todo ); // [삭제] 버튼의 부모 요소 삭제다.  
  }  
}
```

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

버튼 클릭했을 때의 동작 정의하기

```
function removeLocal(todo) {  
  let todos;  
  if (localStorage.getItem('todos') === null) {  
    todos = [];  
  } else {  
    todos = JSON.parse(localStorage.getItem('todos'));  
  }  
  const index = todos.indexOf(todo.children[0].innerText); // 삭제할 할 일의 인덱스  
  todos.splice(index, 1); // index 번째 요소 삭제  
  localStorage.setItem('todos', JSON.stringify(todos)); // 변경된 todos 저장  
}
```

# [실습] 웹 스토리지를 사용한 할 일 목록 만들기

로컬 스토리지에서 해당 내용이 삭제되면서 화면에서도 사라져야 한다

```
function manageTodo(e) {  
  const whichButton = e.target.classList[0]; // 클릭한 부분의 class명 가져오기  
  if(whichButton === 'complete-button') { // [완료] 버튼이면  
    const todo = e.target.parentElement;  
    todo.children[0].classList.toggle('completed'); // 내용 부분에 .completed 클래스 토글  
  } else if(whichButton === 'delete-button') { // [삭제] 버튼이면  
    const todo = e.target.parentElement; // [삭제] 버튼의 부모 요소를 todo에 할당  
    removeLocal(todo ); // [삭제] 버튼의 부모 요소 삭제다.  
    todo.remove()  
  }  
}
```

---

# 지오로케이션 API

---



# 위치 정보 서비스

위치 정보를 악용하면 본인의 의사와 상관없이 위치를 추적할 수 있게 되어 개인의 사생활을 침해할 수도 있다.

→ 사용자가 동의해야만 위치 정보를 사용할 수 있습니다.



지오로케이션 API로 사용자 위치 정보를 다루는 프로그램이면 반드시 보안이 갖춰진 https 프로토콜 환경에서만 사용해야 한다.

# getCurrentPosition( ) 메서드

## geolocation 객체

현재 위치를 알아내고 움직이는 사용자의 위치까지 추적할 수 있는 객체(window.navigator 객체의 자식 객체)

## getCurrentPosition() 메서드

- 현재 위치를 알아내는 함수

**기본형** `getCurrentPosition(successCallback[, errorCallback, options])`

- successCallback: 메서드를 실행해서 위치를 성공적으로 가져왔을 때 실행할 콜백 함수
- errorCallback: 메서드를 실행해서 위치를 가져오지 못했을 때 실행할 콜백 함수. 필수 항목은 아니다.
- options: 위치 확인에 걸리는 시간 제한이나 정확도를 높게 할 것인지에 대한 여부 등 위치 정보를 확인할 때 사용할 옵션. 필수 항목은 아니다.

# position 객체

- `getCurrentPosition()` 메서드를 사용해 가져온 위치 정보를 저장하는 객체
- 주소 정보나 경도/위도 같은 좌표 정보, 가져온 시간 등이 저장된다.
- `position` 객체의 프로퍼티는 모두 읽기 전용.

자주 사용하는 `position` 객체의 프로퍼티

프로퍼티	기능
<code>address.country</code>	주소 중 국가
<code>address.city</code>	주소 중 시
<code>address.postalCode</code>	주소 중 우편번호
<code>address.street</code>	주소 중 거리 이름
<code>coords.latitude</code>	위치의 경도. degree로 표시
<code>coords.longitude</code>	위치의 위도. degree로 표시
<code>coords.speed</code>	이동 중일 경우 사용자의 움직이는 속도. m/s로 표시
<code>timestamp</code>	위치 정보를 가져온 시간

## (예) 현재 위치 가져오기

```
<button id="getLocation">위치 정보 가져오기</button>
<div id="result"></div>
<script>
  const getLocation = document.getElementById('getLocation');
  getLocation.addEventListener('click', function(e) {
    e.preventDefault();
    if (navigator.geolocation) { // 지오로케이션의 지원 여부 체크
      navigator.geolocation.getCurrentPosition(showPosition, errorPosition);
    } else {
      alert('지오로케이션을 지원하지 않습니다.');
```

# watchPosition( ) 메서드

clearWatch() 메서드를 이용해서 위치 확인을 종료할 때까지 지정한 시간마다 계속 현재 위치를 확인한다  
watchPosition() 메서드의 반환값은 정수

**기본형** watchPosition(successCallback[, errorCallback, options])

- successCallback: 메서드를 실행해서 위치를 성공적으로 가져왔을 때 실행할 콜백 함수
- errorCallback: 메서드를 실행해서 위치를 가져오지 못했을 때 실행할 콜백 함수. 필수 항목은 아니다.
- options: 위치 확인에 걸리는 시간 제한이나 정확도를 높게 할 것인지에 대한 여부 등 위치 정보를 확인할 때 사용할 옵션. 필수 항목은 아니다.

# clearWatch( ) 메서드

watchPosition() 메서드에서 반환한 값을 사용해 clearWatch()를 실행 → 위치 확인 중지.

**기본형** clearWatch(id)

id는 watchPosition() 메서드에서 반환한 값