Programming JS

Margit Tennosaar

JSON

JSON

JSON (JavaScript Object Notation) is a lightweight data format used for exchanging data between a server and a client or between different programming environments. While JSON is derived from JavaScript, it is a language-independent format, making it widely used in web development.

Use cases of JSON

- APIs and Web Services – JSON is the standard format for data exchange between clients and servers.
- Configuration Files – Used in settings and application configurations (e.g., package.json in Node.js).
- Data Storage – NoSQL databases like MongoDB use a JSON-like format for storing data.

Benefits of JSON

- Cross-Platform – Can be used with almost any programming language.
- Human-Readable – Simple and easy to understand.
- Lightweight – More compact than XML, reducing network usage.

Objects – Enclosed in curly braces {} and contain key-value pairs.

Arrays – Enclosed in square brackets [] and hold multiple values.

Strings – Must be enclosed in double quotes "".

Numbers, Booleans, and null – Represented the same way as in JavaScript.

```json
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "courses": ["Math", "Science"],
  "address": {
    "city": "New York",
    "zipcode": "10001"
  }
}
```

Converting JavaScript Objects to JSON

When working with JSON in JavaScript, you often need to convert data between JavaScript objects and JSON strings.

```
const person = {
  name: "John Doe",
  age: 30
};

const jsonString = JSON.stringify(person);
console.log(jsonString);
// Output: '{"name":"John Doe","age":30}'
```

Converting JSON Strings to JavaScript Objects

When receiving JSON data from a server, convert it into a JavaScript object using JSON.parse().

```
const jsonString = '{"name":"John Doe","age":30}';

const personObject = JSON.parse(jsonString);
console.log(personObject);
// Output: { name: "John Doe", age: 30 }
```

Storing and Retrieving JSON Data

JSON is often used to store data in LocalStorage or send data in APIs.

```
const settings = { theme: "dark", language: "English" };

// Convert object to JSON and store it
localStorage.setItem("settings", JSON.stringify(settings));
```

Retrieving and Parsing JSON Data

```
const storedSettings = JSON.parse(localStorage.getItem("settings"));
console.log(storedSettings.theme);
// Output: "dark"
```

JSON limitations

JSON Does Not Support Functions or Undefined Values

JSON does not support Date objects, so they must be stored as strings and converted back into Date objects.

LocalStorage

LocalStorage

LocalStorage is a web storage API that allows you to store key-value pairs in the browser. Unlike cookies, LocalStorage persists even when the browser is closed and reopened.

- Persistent Storage – Data remains saved until manually removed.
- Domain-Specific – Data is accessible only from the domain that stored it.
- Storage Limit – Can store approximately 5MB of data.
- Synchronous API – Operations run immediately, which can impact performance for large datasets.

Basic operations

LocalStorage provides four main methods:

setItem(),

getItem(),

removeItem(), and

clear().

Storing data

Use setItem() to save data in LocalStorage.

```
localStorage.setItem('username', 'JohnDoe');
localStorage.setItem('theme', 'dark');
```

If storing objects or arrays, convert them to strings using JSON.stringify().

```
const user = { name: 'John', age: 30 };
localStorage.setItem('user', JSON.stringify(user));
```

Retrieving Data

Use getItem() to retrieve stored data.

```
const username = localStorage.getItem('username');
console.log(username); // Output: JohnDoe
```

If retrieving an object or array, use JSON.parse() to restore its original format.

```
const userData = JSON.parse(localStorage.getItem('user'));
console.log(userData.name); // Output: John
```

Removing Data

To delete a specific item:

```
localStorage.removeItem('username');
```

To remove all stored data:

```
localStorage.clear();
```

Store

```
function savePreferences() {
  const theme = document.querySelector('#theme').value;
  const username = document.querySelector('#username').value;

  localStorage.setItem('theme', theme);
  localStorage.setItem('username', username);

  alert('Preferences saved!');
}
```

Retrieve

```
function loadPreferences() {
  const savedTheme = localStorage.getItem('theme');
  const savedUsername = localStorage.getItem('username');

  if (savedTheme && savedUsername) {
    document.querySelector('#theme').value = savedTheme;
    document.querySelector('#username').value = savedUsername;
    alert(`Welcome back, ${savedUsername}!`);
  }
}
```

Remove

```
function clearPreferences() {
  localStorage.removeItem('theme');
  localStorage.removeItem('username');
  alert('Preferences cleared!');
}
```

Use cases

- Saving user preferences (e.g., theme, language).
- Caching small data to reduce API calls.
- Tracking user progress in games or forms.
- Storing temporary session data between page reloads.

Limitations

- Limited storage size (~5MB).
- Synchronous blocking – Operations pause script execution.
- Security risk – Stored data is accessible via JavaScript, making it vulnerable to XSS attacks.
- Same-origin policy – Data is only accessible on the same domain and protocol.