# Comprendre localStorage en JavaScript

> Le `localStorage` est une API native de JavaScript permettant de stocker des données de manière persistante dans le navigateur.
> Les informations restent disponibles même après fermeture ou actualisation de la page tant qu'elles ne sont pas explicitement supprimées.

## 1. Syntaxe et vocabulaire de localStorage

> Voici une explication détaillée des termes et concepts clés associés au fonctionnement de `localStorage` qui vous aideront à mieux comprendre et utiliser l'API.

### 1.1. Clé

Chaque donnée dans le `localStorage` est stockée sous une clé qui agit comme un identifiant unique. La clé doit être une chaîne de caractères.

#### 1.1.1. Exemple :

```javascript
localStorage.setItem('utilisateur', 'Jean');
```

Dans cet exemple, `utilisateur` est la clé qui permet de retrouver ou manipuler la valeur `'Jean'` .

### 1.2. Valeur

> La valeur est ce qui est associé à une clé dans le `localStorage`.
>
> Par défaut, elle est toujours stockée sous forme de chaîne de caractères, même si vous y ajoutez un nombre, un objet ou un tableau.
> Ces derniers devront être convertis (voir sérialiser ci-dessous).

#### 1.2.1. Exemple rapide :

```javascript
localStorage.setItem('age', 30); // La valeur 30 sera convertie en "30" (string).
const age = localStorage.getItem('age');
console.log(typeof age); // Affiche : string
```

## 1.3. Persistant

> Une propriété importante du `localStorage` est sa *persistance*.
>
> Les données ajoutées au `localStorage` ne sont pas supprimées, sauf si elles sont explicitement effacées à l'aide des méthodes appropriées (`removeItem`, `clear`).

Elles restent disponibles :

- Après le rechargement de la page.
- Après la fermeture et la réouverture du navigateur.
- Tant que l'utilisateur n'efface pas les données de manière manuelle (via les outils du navigateur ou via le JavaScript).

*Exemple pratique :*

```javascript
localStorage.setItem('theme', 'clair');
// Même après avoir fermé et rouvert le navigateur :
console.log(localStorage.getItem('theme')); // Récupère "clair"
```

## 1.4. Sérialiser

Le `localStorage` ne sait manipuler que des chaînes de caractères. Lorsque vous devez stocker des données complexes comme des objets ou des tableaux, il faut les transformer en chaîne de caractères grâce au processus de sérialisation.

### 1.4.1. Exemple :

> Sérialiser un objet (le transformer en chaîne JSON lisible) :

```javascript
const utilisateur = { nom: 'Alice', role: 'Admin' };
localStorage.setItem('utilisateur', JSON.stringify(utilisateur));
```

> Récupérer les données et les reconstruire en objet :

```javascript
const recupere = JSON.parse(localStorage.getItem('utilisateur'));
console.log(recupere.nom); // Affiche : Alice
```

## 1.5. JSON

`JSON` (JavaScript Object Notation) est un format standard utilisé pour stocker et échanger des données.

Le `localStorage` utilise largement ce format pour traiter les objets et les tableaux grâce aux fonctions :

- `JSON.stringify()` : transforme un objet ou un tableau en chaîne JSON.
- `JSON.parse()` : transforme une chaîne JSON en un objet ou un tableau utilisable

### 1.5.1. Exemple :

```javascript
// Enregistrement :
const fruits = ['pomme', 'banane'];
localStorage.setItem('fruits', JSON.stringify(fruits));

// Lecture :
const listeFruits = JSON.parse(localStorage.getItem('fruits'));
console.log(listeFruits); // Affiche : ['pomme', 'banane']
```

## 1.6. SessionStorage vs LocalStorage

Bien que souvent comparés, les termes `localStorage` et `sessionStorage` diffèrent dans leur persistance.

- localStorage : Les données sont persistantes sur le long terme.
- sessionStorage : Les données existent uniquement pendant la durée de la session du navigateur (elles disparaissent une fois le navigateur fermé).

## 1.7. Taille maximale

Chaque navigateur impose une *limite* sur la quantité de données pouvant être stockées (généralement 5 Mo).

Tenter de dépasser cette taille génèrera une erreur.

### 1.7.1. Exemple d'erreur :

```javascript
try {
  localStorage.setItem('cléLourde', 'x'.repeat(1024 * 1024 * 6)); // ~6 Mo
} catch (e) {
  console.error("Espace insuffisant !"); // S'exécute si la limite est dépassée
}
```

## 1.8. Multi-domaine

> Les données dans le `localStorage` sont associées au domaine (ou sous-domaine) utilisé.
>
> Cela garantit qu'un site A n'accède pas aux données du site B, renforçant ainsi l'isolation entre les applications.

### 1.8.1. Exemple :

- `https://exemple.com` aura un stockage séparé.
- `https://admin.exemple.com` ne pourra jamais accéder aux données créées par `https://exemple.com` .

---

## Sérialisation de tableaux dans localStorage

> Les tableaux doivent être convertis en chaînes JSON avant d'être stockés.
> De même, ils doivent être reconvertis en tableau lors de la récupération.

## 2.1. Sauvegarder un tableau

> Sérialisation d'un tableau en chaîne JSON à l'aide de `JSON.stringify()` :

```javascript
const fruits = ['pomme', 'banane', 'orange'];
localStorage.setItem('fruits', JSON.stringify(fruits));
```

## 2.2. Récupérer un tableau

Récupération et désérialisation de données stockées sous forme de tableau :

```javascript
const fruits = JSON.parse(localStorage.getItem('fruits'));
console.log(fruits); // Affiche : ['pomme', 'banane', 'orange']
```

## 2.3. Modifier un élément d'un tableau

Étape intermédiaire pour mettre à jour une des valeurs d'un tableau existant :

```javascript
const fruits = JSON.parse(localStorage.getItem('fruits')); // Récupérer le tableau
fruits.push('kiwi'); // Ajouter un nouvel élément
localStorage.setItem('fruits', JSON.stringify(fruits)); // Re-sauvegarder
```

---

## Sérialisation d'objets dans localStorage

> Les objets suivent un processus similaire aux tableaux pour leur sérialisation/désérialisation afin d'être gérés dans le `localStorage`.

### 3.1. Enregistrer un objet

Utilisation de `JSON.stringify()` pour stocker des propriétés complexes :

```javascript
const utilisateur = { nom: 'Alice', role: 'Admin' };
localStorage.setItem('utilisateur', JSON.stringify(utilisateur));
```

### 3.2. Récupérer un objet

Convertir la chaîne JSON récupérée en un objet utilisable avec `JSON.parse()` :

```javascript
const utilisateur = JSON.parse(localStorage.getItem('utilisateur'));
console.log(utilisateur.nom); // Affiche : Alice
```

### 3.3. Mises à jour sur un objet

Pour modifier une propriété d'un objet stocké dans le `localStorage` :

```javascript
const utilisateur = JSON.parse(localStorage.getItem('utilisateur'));
// Récupérer l'objet
utilisateur.role = 'Utilisateur'; // Modifier une propriété

localStorage.setItem('utilisateur', JSON.stringify(utilisateur));
// Sauvegarder l'objet mis à jour
```

---

## À RETENIR

| Terme clé | Définition et Utilisation |
|---|---|
| localStorage | **Stocker** des données sous forme clé-valeur dans le navigateur de manière persistante. |
| Clé / Valeur | La clé est **toujours une chaîne**, la valeur est stockée **comme une chaîne** également. |
| Persistant | Données conservées **après rechargement** ou fermeture du navigateur, sauf suppression explicite. |
| Sérialiser / Désérialiser | Convertir en chaîne JSON ( `JSON.stringify` ) ou reconstruire ( `JSON.parse` ) des données complexes. |
| JSON | Format utilisé pour **échanger** et **manipuler** des objets ou tableaux via localStorage. |
| Taille Maximale | Généralement limitée à ~5 Mo, dépend des navigateurs. |
| Sécurité | Les données ne sont **pas chiffrées**, ne pas stocker d'**informations sensibles**. |
| SessionStorage | Diffère par sa durée de vie **limitée** à la session du navigateur. |