Browser storage

As web developers, it's important to understand how to store data locally in a user's browser to enhance the user experience and functionality of our web applications. We'll look into the different types of browser storage available, their limitations, and how to use them in your web applications.

Caveats:

It's important to note that browser storage is only available in the user's **current browser**. For example, if we're using Google Chrome as our browser and we save information to it, that information is only available in Google Chrome on the device we are currently using.

This means that if we switch to a different browser or device, we don't have access to that information. This is important to keep in mind as it's easy for users to clear their browser storage, move to a different device, or switch to a different browser, and in that case, the stored data will not be accessible.

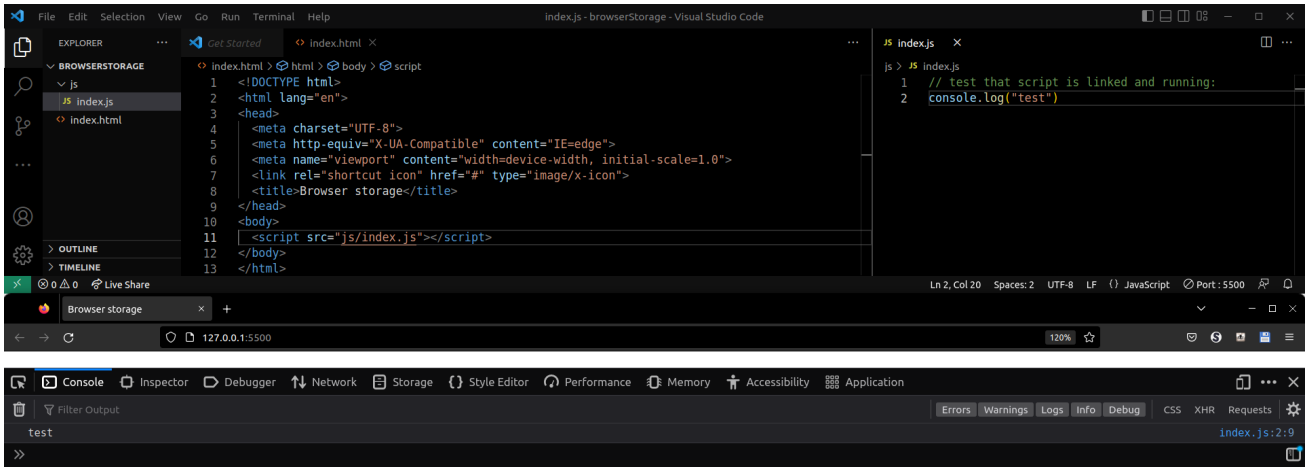
Types of browser storage:

There are three different types of browser storage: **cookies**, **local storage**, and **session storage**.

- **Local storage** allows us to store **10 megabytes of storage for each item** inside of local storage. This is plenty of storage space and if we exceed this, we should probably rethink how we are storing the information. Data in local storage **never expires**.
  Typical use cases include:
    - Saving user preferences on a website (such as preferred color scheme)
    - Caching data for offline use
    - Storing form data to be auto-filled next time the user visits the website
    - Saving the state of a web application between page refreshes or browser restarts
- **Session storage** allows us to store around **5 megabytes of storage space** which is again quite large and more than we will need for most use cases. Data in session storage **expires when the user closes the browser**.
  Typical use cases include:
    - Storing temporary data for a single session, such as shopping cart items in an e-commerce website
    - Storing temporary data for a single tab, such as form data in a multi-step form
    - Storing data for an authenticated user session
- **Cookies** only allow us to store **4 kilobytes which is a very small** amount of information, but cookies generally only contain a small amount of information so this is not something we will run into as an issue. We can set an expiration for the cookies by **using the "expires" attribute**.
  Typical use cases include:
    - Storing user login information to keep the user logged in on future visits
    - Tracking user preferences and behavior for targeted advertising
    - Storing session IDs for server-side session management
    - Saving the state of a web application between page refreshes or browser restarts

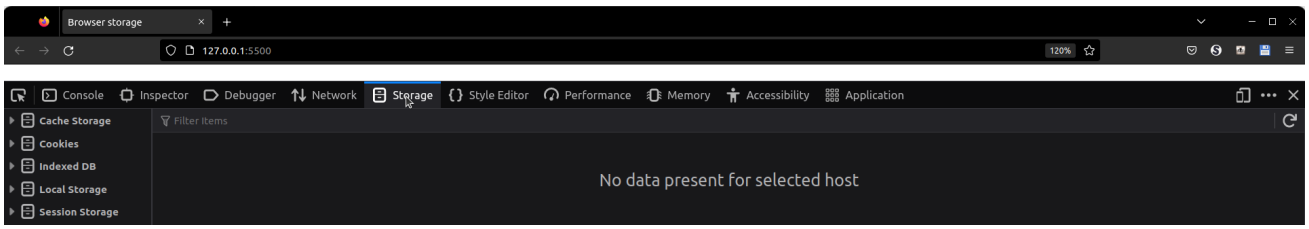
Chances are you've probably heard of cookies before, but in reality, these are by far the least useful version of browser storage. Local storage and session storage are much easier to use and you will probably find them to be the most useful as well.

Before we dive deeper into how we can use these storage options, open VSCode and start an empty project, with some barebone index.html, and link to an empty javascript file. Then start live-server, so that you have the page open in a browser (and then open the developer tools), for example like this:

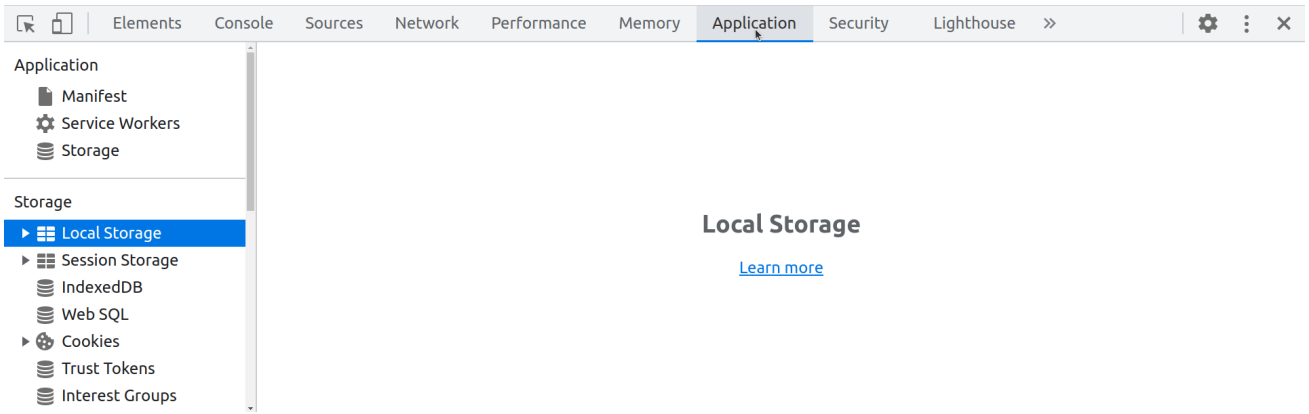


In the example above i'm using the Firefox browser (which is also the browser i'd recommend for web-design, due to its developer tools being more useful for us developers than for example Chrome).

To view browser storage, in **Firefox** just click the storage tab:



In **Chrome**, click Application:



Then on the left-hand side of the dev-tools you'll see the various storage types.

Local Storage:

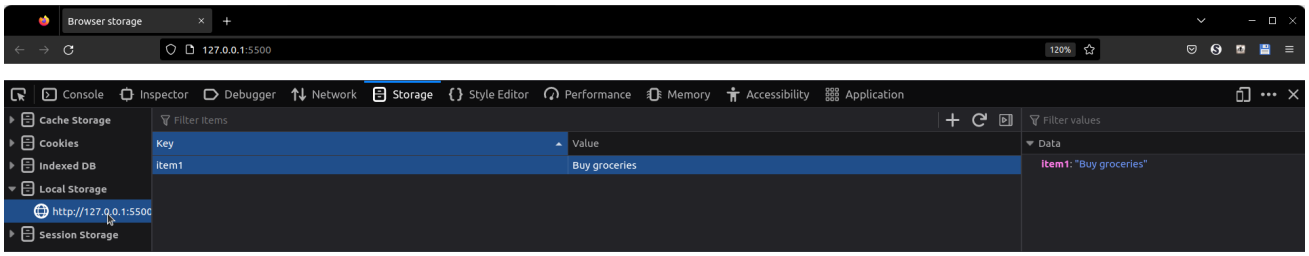
To use local storage, we use an object called: **localStorage**. This object is part of the **global window object**, and since its a global object we don't need to write **window.localStorage**, we can just write **localStorage**. Just like we can write **console.log** instead of **window.console.log**.

To demonstrate how to use local storage, let's consider a simple example of a to-do list application. In this application, we want to store the list of to-do items in local storage so that they are not lost when the user closes the browser or refreshes the page.

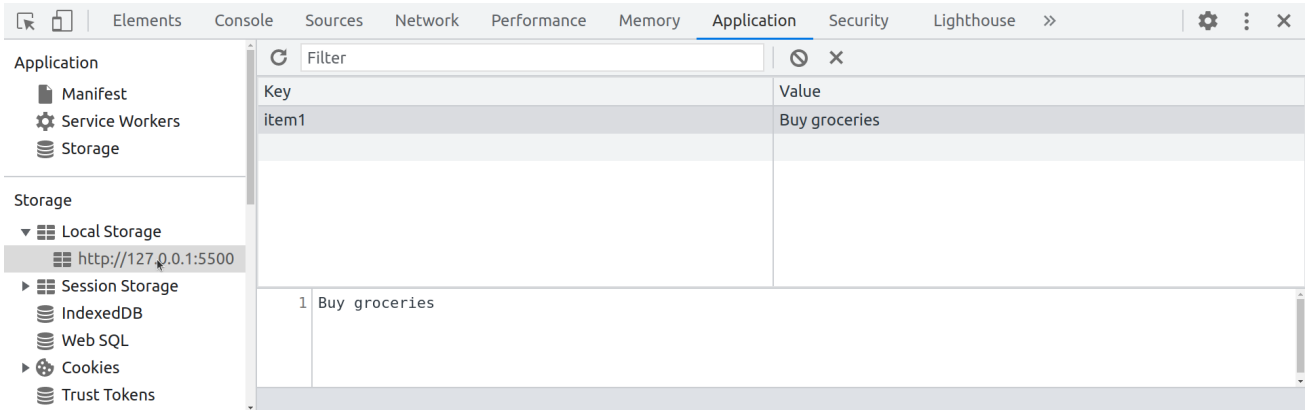
To save an item to local storage, we use the **setItem() method**, which takes **two parameters**: a **key** and a **value**. For example, we can save a to-do item with the key "item1" and value "Buy groceries" as follows:

```
localStorage.setItem("item1", "Buy groceries");
```

After we save the script with that code, we can immediately see the local storage being updated in the browser, we find that under the Local Storage tab, first how it looks like in Firefox::



And in Chrome:



Notice how in the dev-tools, under Local Storage there is a url, mine is http://127.0.0.1:5500 (which is the live-server that VSCode gives us when we start a live-server). This is because Local Storage works on a per url basis, so all my projects that run on the url http://127.0.0.1:5500 will share the exact same Local Storage. This is something we'll talk about more later on, for now let's continue to see how we can also update existing items and remove them from local storage.

- We can update data in local storage by using the same **localStorage.setItem** method.

```
localStorage.setItem("item1", "Prepare dinner");
```

- To retrieve the data from local storage, we use the getItem() method, which takes the key as a parameter and returns the corresponding value. For example, we can retrieve the to-do item with the key "item1" as follows:

```
console.log(localStorage.getItem("item1")); // Prepare dinner
```

- We can also remove items from local storage using the removeItem() method, which takes the key as a parameter. For example, we can remove the to-do item with the key "item1" as follows:

```
localStorage.removeItem("item1");
```

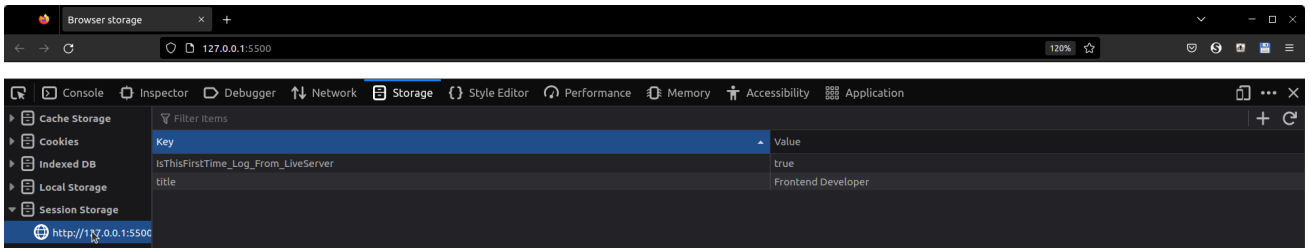
Session Storage:

Session storage is similar to local storage, but the data is only available for the current browsing session and is deleted when the user closes the browser or when the session ends. This is useful for storing temporary data that needs to persist only for the duration of the session. The storage limit for session storage is around 5 megabytes, which is more than enough for most use cases.

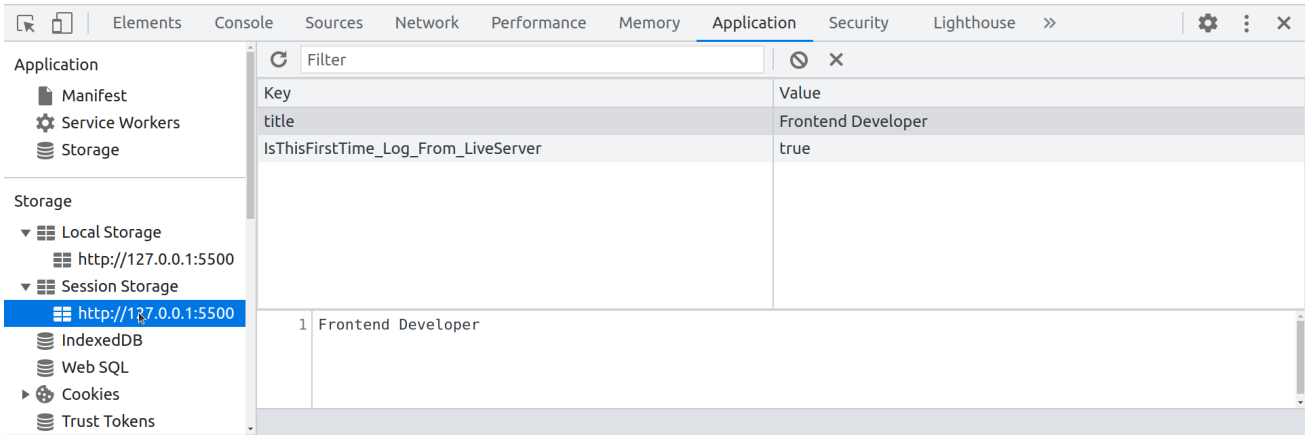
To use session storage, we use the same setItem(), getItem(), and removeItem() methods as we do for local storage. For example, we can save a user's title to session storage as follows:

```
sessionStorage.setItem("title", "Frontend Developer");
```

After saving the script, we can see in the dev-tools the following data under Session Storage tab (in Firefox):



The same in Chrome:



And retrieve it later as follows:

```
console.log(sessionStorage.getItem("title")); // Frontend Developer
```

We can update data in session storage by using the **sessionStorage.setItem** method.

```
sessionStorage.setItem("title", "Fullstack Developer");
```

We can remove items from session storage by using the **sessionStorage.removeItem** method.

```
sessionStorage.removeItem("title");
```

Complete list of all Local Storage, and Session Storage **methods** / **properties**:

- `length` - returns the number of items in the storage
- `clear()` - removes all items in the storage
- `removeItem(key)` - removes an item in the storage with the given key
- `setItem(key, value)` - updates (or creates if key does not exist) an item with the given the key and the given value
- `key(index)` - return the name of the item at the given index (number)

Cookies:

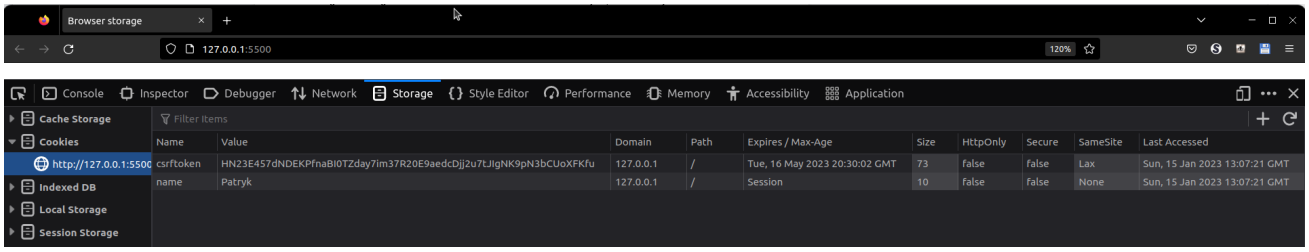
Cookies, in contrast to local storage / session storage, are not as straight-foreward to work with. The interface for reading and writing cookies is an attribute under the document object, conveniently called **document.cookie.**

Cookies are small text files that are stored on the user's computer by the browser. They are mainly used to remember user preferences, login information, and to track user behavior on a website. The storage limit for cookies is 4 kilobytes, which is quite small, but cookies are mainly used to store small pieces of information.

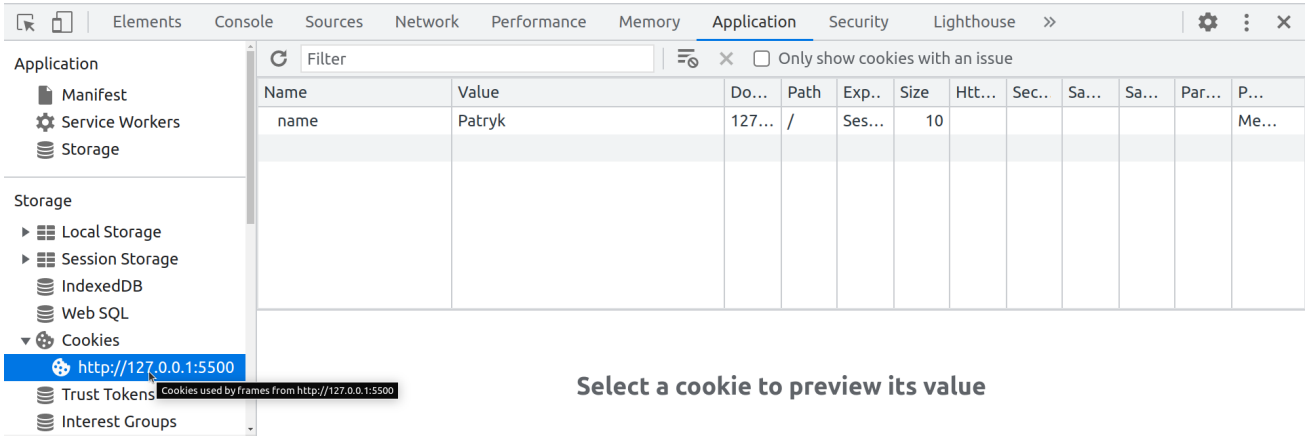
To create a cookie, we need to set the **document.cookie** attribute to a key-value pair. For example, we can create a cookie for a user's name as follows:

```
document.cookie = "name=Patryk";
```

After saving the script, we can see in the dev-tools the following data under Cookies tab (in Firefox):



The same in Chrome:



To retrieve a cookie, we can use the same document.cookie attribute and access the data.

```
console.log(document.cookie); // name=Patryk
```

There are also users' privacy concerns at hand with cookies, you need to ask the users for permission to store/use cookies.

In conclusion, browser storage is a powerful tool for storing data locally in a user's browser. By understanding the different types of storage available, their limitations, and how to use them, we can enhance the user experience and functionality of our web applications.

Before we proceed to implement local storage in our existing to-do list application, let's first familiarize ourselves with JSON (JavaScript Object Notation). JSON is a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is a common format for storing and transferring data on the web and in JavaScript applications.

In our existing to-do list application, we use an array of to-do item objects to store the to-do items. To save the entire to-do list array to local storage, we can use JSON.stringify() method to convert the array to a JSON string, and then use the setItem() method to save it to local storage under a key of "todoList".

```
let todoList = [
 { name: "todoItem1", id: 1, isCompleted: false },
 { name: "todoItem2", id: 2, isCompleted: true }
];
localStorage.setItem("todoList", JSON.stringify(todoList));
```

Retrieving the to-do list array is done by using the getItem() method to retrieve the JSON string and then using the JSON.parse() method to convert it back to an array.

```
let todoList = JSON.parse(localStorage.getItem("todoList"));
```

By storing the entire to-do list array in local storage, we can easily add, remove, and update to-do items in the array and have them persist across different sessions.

Why not store each todo item under its own key in local storage ?

If we'd store each to-do item under a separate key in local storage, it would lead to a large number of keys being created and maintained in the browser's storage. This would increase the memory overhead and make it difficult to manage the data.

On the other hand, if we store the entire to-do list array under a single key, it would only require a single key to be created and maintained, which greatly reduces the memory overhead and makes it much easier to manage the data.

Additionally, when we need to retrieve the data, we can retrieve it all at once by using a single key, instead of having to retrieve each item separately using multiple keys.

Also, when we need to share the data with other parts of the application or other applications, it's easier to share a single piece of data than multiple pieces of data.

In summary, storing the entire to-do list array under a single key in local storage allows for more efficient and simplified data management and sharing.

You can find more information about browser storage at Mozilla Developer Network

Below is a multiple choice quiz to help you self-evaluate your understanding and solidify your knowledge of the different types of browser storage and how to use them effectively.

1. What is the limit of storage space for Local Storage?
   - ☐ a) 10KB
   - ☐ b) 10MB
   - ☐ c) 100MB
   - ☐ d) 1GB

2. Which of the following are the types of browser storage?
   - ☐ a) Local Storage
   - ☐ b) Session Storage
   - ☐ c) Cookies

3. How do you retrieve a value from Local Storage?
   - ☐ a) localStorage.get("key")
   - ☐ b) localStorage.value("key")
   - ☐ c) localStorage.getItem("key")
   - ☐ d) localStorage.valueOf("key")

4. How do you remove a value from Local Storage?
   - ☐ a) localStorage.remove("key")
   - ☐ b) localStorage.delete("key")
   - ☐ c) localStorage.removeItem("key")
   - ☐ d) localStorage.deleteItem("key")

5. What is the main difference between Local Storage and Session Storage?
   - ☐ a) Local Storage has a larger storage limit
   - ☐ b) Session Storage data is automatically deleted when the browser is closed
   - ☐ c) Local Storage data is automatically deleted when the browser is closed
   - ☐ d) Session Storage has a larger storage limit

6. If you want to store an array of objects in Local Storage, what is the most efficient way to do this?
   - ☐ a) Saving each object as a separate key in Local Storage
   - ☐ b) Saving the array as a JSON string under a single key in Local Storage

7. What are some use cases for Session Storage?
   - ☐ a) Storing temporary data for a single session
   - ☐ b) Storing data for an authenticated user session
   - ☐ c) Storing temporary data for a single tab
   - ☐ d) Storing data that needs to be shared across multiple tabs or windows of the same browser.

8. What are some use cases for Cookies?
   - ☐ a) Storing user login information
   - ☐ b) Tracking user preferences and behavior

- [ ] c) Storing session IDs for server-side session management
- [ ] d) Remembering user's shopping cart items in an e-commerce website.

And here is the same quiz in a format that's compatible with your quiz app that we worked on previously:

```
const quizBrowserStorageQuestions = [
{
 question: "What is the limit of storage space for Local Storage?",
 choices: ["10KB", "10MB", "100MB", "1GB"],
 correctAnswers: ["10MB"],
 userAnswer: []
},
{
 question: "Which of the following are the types of browser storage?",
 choices: ["Local Storage", "Session Storage", "Cookies", "None of the
above"],
 correctAnswers: ["Local Storage", "Session Storage", "Cookies"],
 userAnswer: []
},
{
 question: "How do you retrieve a value from Local Storage?",
 choices: ["localStorage.get('key')", "localStorage.value('key')",
"localStorage.getItem('key')", "localStorage.valueOf('key')"],
 correctAnswers: ["localStorage.getItem('key')"],
 userAnswer: []
},
{
 question: "How do you remove a value from Local Storage?",
 choices: ["localStorage.remove('key')", "localStorage.delete('key')",
"localStorage.removeItem('key')", "localStorage.deleteItem('key')"],
 correctAnswers: ["localStorage.removeItem('key')"],
 userAnswer: []
},
{
 question: "What is the main difference between Local Storage and Session
Storage?",
 choices: [
   "Local Storage has a larger storage limit",
   "Session Storage data is automatically deleted when the browser is
closed",
   "Local Storage data is automatically deleted when the browser is closed",
   "Session Storage has a larger storage limit"
 ],
 correctAnswers: ["Session Storage data is automatically deleted when the
browser is closed"],
 userAnswer: []
```

```
},
{
 question: "If you want to store an array of objects in Local Storage, what
is the most efficient way to do this?",
 choices: [
   "Saving each object as a separate key in Local Storage",
   "Saving the array as a JSON string under a single key in Local Storage"
 ],
 correctAnswers: [
   "Saving the array as a JSON string under a single key in Local Storage"
 ],
 userAnswer: []
}];
```