Storing Data in JavaScript

JSON Format

JSON is one of the newer data *storage methods* in JavaScript.

It has been heavily used only in the past few years.

It stands for *JavaScript Object Notation* and is a very lightweight format for exchanging data both on server and on client.

JSON Format

It is not only easy to read and write for humans, but it is also easy to read and write for machines.

All modern programming languages support JSON in some way.

This makes it great when we have to integrate with other systems that may not be running on same platform.

JSON Format

```
var favoriteSandwiches = {
   "breakfast" : [
      {
         "name": "Egg, Sausage and Cheese",
         "bread": "English Muffin"
      },
      {
         "name": "Egg Whites on Flatbread",
         "bread": "Flatbread"
      }
   ],
   ...
}
```

JSON Format

```
var favoriteSandwiches = {
    ...
    "lunch" : [
        {
            "name": "Turkey Club",
            "bread": "Wheat Bread"
        },
        {
            "name": "Grilled Cheese",
            "bread": "White Bread"
        }
    ],
    ...
}
```

JSON Format

```
var favoriteSandwiches = {
    ...
    "dinner" : [
        ...
    ]
};

/* go to dinner and get name of first item */
alert(favoriteSandwiches.dinner[0].name); // Meatball
```

Benefits of using JSON

JSON is fast, lightweight, and can easily travel from one domain to another.

JSON also matches data models from other programming languages such as JavaScript, whereas XML does not.

It's great for Ajax, and it's great for general data storage.

Benefits of using JSON

JSON is used with a lot of external services to transfer data because it can be consumed cross-domain.

Before JSON was popular, XML was the data format of choice if we needed to consume data from an outside source.

But we weren't able to directly access XML cross-domain with JavaScript without passing it through a local proxy.

Using an API

Many web services will offer an API (stands for *Application Programming Interface*) to developers to build and expand on their codebase.

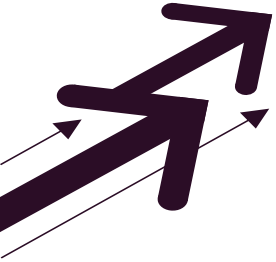
It is a way for a site to safely give us access to information they store in their databases for our own personal use.

Using an API

Instead of the search results coming back in a well-designed search result interface, the API results in a JSON format for us to parse through.

Most modern-day APIs work this way: we make a call and we get back data in JSON format.

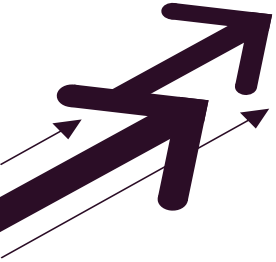
APIs are probably most common use of JSON today because the JSON format makes it consumable by any code base.



Web Storage in HTML5

User interactions are what we do with JavaScript: it creates a behavior layer.

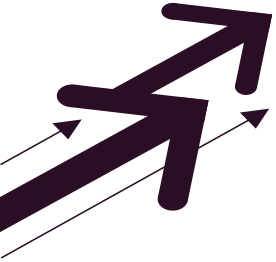
With that in mind, one next step is to start using storage as a way to interact with the user.



Web Storage in HTML5

In the past we've used cookies to save user-generated data, but cookies are hacky, a pain to deal with, and insecure.

The main problem with cookies is that they got such a bad rap for being riddled with tracking information and malware that people often browse with them turned off.

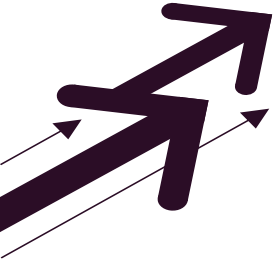


Web Storage in HTML5

So a new standard for *Web Storage* was created in HTML5: *Local Storage* and *Session Storage*.

These two new *Storage APIs* are already changing the way we interact with user-initiated stored data.

It's a powerful tool for creating a better and more personalized user experience in the browser.

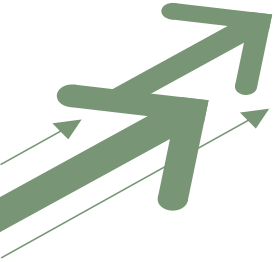


Web Storage in HTML5

In HTML5 we have access to two new storage objects: **localStorage** and **sessionStorage**.

These objects are paired with **setItem**, **getItem**, and **removeItem** methods.

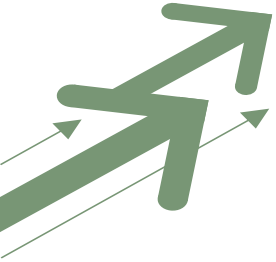
Calling these methods with **localStorage** will store the data locally with the user.



Local Storage

To manually save an object to **localStorage** we need to use the **setItem** method that it takes these arguments:

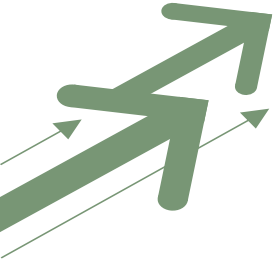
- The first is the equivalent of a variable label, or what we want to call the data being stored.
- The second argument is the data itself, which must be a string.



Local Storage

This code shows how we would manually save an object called **"favoriteSandwich"** with a value of **"Meatball"**.

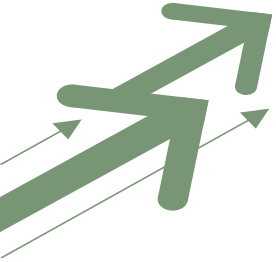
```
/* set localStorage */
localStorage.setItem("favoriteSandwich", "Meatball");
```



Local Storage

After setting an item, it's only natural that we would want, at some point, to retrieve that same item.

For this we use the **getItem** method and it takes a single argument, which is the name of item we previously set.

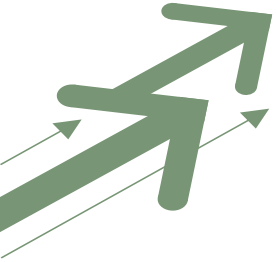


Local Storage

This code shows how to retrieve the **"favoriteSandwich"** data that was set before by using **setItem** method.

```
/* retrieve stored data */
var sandwich =
   localStorage.getItem("favoriteSandwich"); // "Meatball"

/* prove it was set */
alert(sandwich);
```

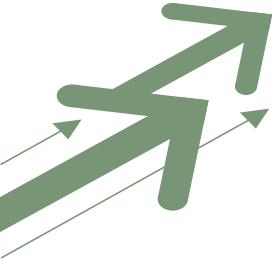


Local Storage

The method in *Local Storage* to help us clean up the data it is called **removeItem**.

This method will delete the item stored locally with the user.

It takes one argument, just like its companion method, **getItem**.



Local Storage

This code shows how to remove a locally stored item.

```
/* delete stored data */
localStorage.removeItem("favoriteSandwich");
```

Session Storage VS Local Storage

Both **localStorage** and **sessionStorage** use the same syntax and methods.

The only is that **localStorage** will remain active until explicitly removed by us via the browser preferences.

Because **sessionStorage** is a *session-based storage method*, this will remove itself at the end of each browser session (when the browser is closed).

Storing Chunks of Data with JSON

Web Storage API currently can take only strings for data.

This can be problematic when we need to store multiple items.

We can get around this limitation by storing our data in a *JSON object*.

Using the **JSON.stringify** method, we can convert entire object to a string that can be stored using *Web Storage*.

Storing Chunks of Data with JSON

This code shows how we might take our entire *JSON object* that was created before and save it locally.

```
/* stringify the JSON object first */
var stringObject = JSON.stringify(favoriteSandwiches);

/* add the string object to localStorage */
localStorage.setItem("favoriteSandwiches", stringObject);
```

Storing Chunks of Data with JSON

This *JSON object* is now a string, and the individual items in this object can't be accessed efficiently.

When we access this item it needs to be reconverted from a string, back into a *JSON object*.

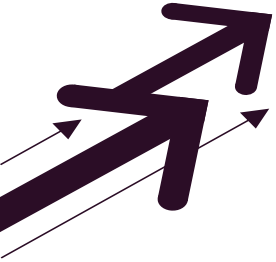
Storing Chunks of Data with JSON

This code shows us how to use **JSON.parse** to convert this string back into a workable *JSON format*.

```
/* get the locally stored data */
var storedItem = localStorage.getItem("favoriteSandwiches");

/* convert it from a string, back into a JSON object */
var convertObject = JSON.parse(storedItem);

/* prove it worked */
alert(convertObject.breakfast[0].name); // Egg, Sausage and Cheese
```

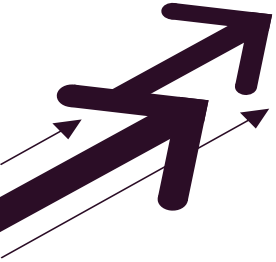


Using Web Storage Responsibly

Web Storage can be not supported every browsers yet.

This, coupled with a general desire to create a fluid user experience no matter what browser a user is in, brings us back to something called *feature detection*.

It will be a main theme whenever we're coding for these new standards brought forth by HTML5.

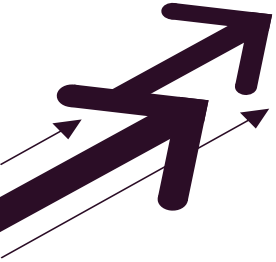


Using Web Storage Responsibly

The main idea is that **localStorage** and **sessionStorage** are *features of advanced browsers*.

We can detect for the presence of these *advanced features* and then code for or against it.

If a *feature* doesn't exist, we can use an alternative method that may be an older or that might not perform as well, but it still gets the job done.



Using Web Storage Responsibly

This code shows us how we might use *feature detection* with **localStorage**.

```
/* check for localStorage support */
if(typeof(localStorage) === "undefined") {
   // localStorage is not supported, maybe use cookies?

} else {
   // localStorage is supported, use it and be well.
   // add your localStorage code in here
}
```