JSON Learning Objectives

The objective of this lessonis to familiarize you with the JSON format and how to serialize to and deserialize from that format.

The learning objectivesfor this lesson are that you can:

1. Identify and generate valid JSON-formatted strings
2. Use `JSON.parse`to deserialize JSON-formatted strings
3. Use `JSON.stringify`to serialize JavaScript objects
4. Correctly identify the definition of "deserialize"
5. Correctly identify the definition of "serialize"

This lesson is relevantbecause JSON is the *lingua franca* of data interchange.

Storage Lesson Learning Objectives

Below is a complete list of the terminal learning objectives for this lesson. When you complete this lesson, you should be able to perform each of the following objectives. These objectives capture how you may be evaluated on the assessment for this lesson.

1. Write JavaScript to store the value "I <3 falafel" with the key "eatz" in the browser's local storage.
2. Write JavaScript to read the value stored in local storage for the key "paper-trail".

Cookies and Web Storage

As we've learned in previous sections, most data on the Web is stored in a database on a server, and we use the browser to retrieve this data. However, sometimes data is stored locally for the purposes of persisting throughout an entire session or until a specified expiration date.

In this reading, we'll go over using **cookies**to store data versus using the**Web Storage API** and the use cases for each storage method.

Cookies

Cookies have been around forever, and they are still a widely used method to store information about a site's users.

What is a cookie?

A cookie is a small file stored on a user's computer that holds a bite-sized amount of data, under 4KB. Cookies are included with HTTP requests. The server sends the data to a browser, where it's typically stored and then sent back to the server on the next request.

What are cookies used for?

Cookies are used to store stateful information about a user, such as their personal information, their browser habits or history, or form input information they have filled out. A common use case for cookies is storing a *session cookie*on user login/validation. Session cookies are lost once the browser window is closed. To make sure the cookie persists beyond the end of the session, you could set up a *persistent cookie*with a specified expiration date. A use case for a persistent cookie is an e-commerce website that tracks a user's browsing or buying habits.

How to create a cookie in Javascript:

As we've previously covered, the `document`interface represents the web page loaded in a user's browser. Since cookies are stored on a user's browser, it makes sense that the `document`object also allows us to get/set cookies on a user's browser:

```
const firstCookie = "favoriteCat=million";
document.cookie = firstCookie;
const secondCookie = "favoriteDog=bambi";
document.cookie = secondCookie;
document.cookie; // Returns "favoriteCat=million; favoriteDog=bambi"
```

Using the following syntax will create a new cookie:

```
document.cookie = aNewCookieHere;
```

If you want to set a second cookie, you would assign a new key value pair using the same syntax a second time. Make sure to set the cookie to a string formatted like a key-value pair:

```
const firstCookie = "favoriteCat=million";
document.cookie = firstCookie;
document.cookie; // Returns "favoriteCat=million"
```

Formatting your string like we do in the `firstCookie`variable above sets the cookie `value`with a defined key, known as the cookie's `name`, instead of an empty `name`. Refer to the MDN docs on Document.cookiefor more examples.

You can view all the cookies a website is storing about you by using the Developer Tools. On **Google Chrome**, see the **Application tab**, and on**Firefox**, see the **Storage tab**.

Deleting a cookie:

We can delete our own cookies using JavaScript by setting a cookie's expiration date to a date in the past, causing them to expire:

```
const firstCookie = "favoriteCat=million";
document.cookie = firstCookie;
document.cookie; // Returns "favoriteCat=million"

// specify the cookies "name" (the key) with an "=" and set the  expiration
// date to the past
document.cookie = "favoriteCat=; expires = Thu, 01 Jan 1970 00:00:00 GMT";
document.cookie; // ""
```

We can also delete cookies using the Developer Tools!

Navigate to a website, such as Amazon, and add an item to your cart. Open up the Developer Tools in your browser and delete all the cookies. In Chrome, you can delete cookies by highlighting a cookie and clicking the delete button. In Firefox, you can right-click and delete a cookie. If you've deleted all the cookies in your Amazon cart, and you refresh the page, you should notice your cart is now empty.

Web Storage API

Cookies used to be the only way to store data in the browser, but with HTML5 developers gained access to the Web Storage API, which includes**localStorage**and **Session Storage**. Here are the differences between the two, according to MDN:

`sessionStorage`:

- Stores data only for a *session*, or until the browser window or tab is closed
- Never transfers data to the server
- Has a storage limit of 5MB (much larger than a cookie)

The following example from MDNshows how we can use sessionStorage to autosave the contents of a text field and restore the contents of that text field if the browser is accidentally refreshed.

```
// Get the text field that we're going to track
let field = document.getElementById("field");

// See if we have an autosave value
// (this will only happen if the page is accidentally refreshed)
if (sessionStorage.getItem("autosave")) {
  // Restore the contents of the text field
  field.value = sessionStorage.getItem("autosave");
}

// Listen for changes in the text field
field.addEventListener("change", function () {
  // And save the results into the session storage object
  sessionStorage.setItem("autosave", field.value);
});
```

`localStorage`:

- Stores data with no expiration date and is deleted when clearing the browser cache
- Has the maximum storage limit in the browser (much larger than a cookie)

Like with `sessionStorage`, we can use the `getItem()`and `setItem()`methods to retrieve and set `localStorage`data. The following example from MDNwill:

- Check whether `localStorage`contains a data item called `bgcolor`using`getItem()`.
- If `localStorage`contains `bgcolor`, run a function called `setStyles()`that grabs the data items using `Storage.getItem()`and use those values to update page styles.
- If it doesn't, run a function called `populateStorage()`, which uses`Storage.setItem()`to set the item values, then run `setStyles()`.

```
if (!localStorage.getItem("bgcolor")) {
  populateStorage();
}
setStyles();

const populateStorage = () => {
  localStorage.setItem("bgcolor", document.getElementById("bgcolor").value);
  localStorage.setItem("font", document.getElementById("font").value);
  localStorage.setItem("image", document.getElementById("image").value);
};

const setStyles = () => {
  var currentColor = localStorage.getItem("bgcolor");
  var currentFont = localStorage.getItem("font");
  var currentImage = localStorage.getItem("image");

  document.getElementById("bgcolor").value = currentColor;
  document.getElementById("font").value = currentFont;
  document.getElementById("image").value = currentImage;

  htmlElem.style.backgroundColor = "#" + currentColor;
  pElem.style.fontFamily = currentFont;
  imgElem.setAttribute("src", currentImage);
};
```

When would we use the Web Storage API?

Since web storage can store more data than cookies, it's ideal for storing multiple key-value pairs. Like with cookies, this data can be saved only as a string. With localStorage, the data is stored locally on a user's machine, meaning that it can only be accessed client-side. This differs from cookies which can be read both server-side and client-side.

There are a few common use cases for Web storage. One is storing information about a shopping cart and the products in a user's cart. Another is saving input data on forms. You could also use Web storage to store information about the user, such as their preferences or their buying habits. While we would normally use a cookie to store a user's ID or a session ID after login, we could use localStorage to store extra information about the user.

You can view what's in local or session storage by using the Developer Tools. On**Google Chrome**, see the **Application tab**, and on **Firefox**, see the**Storage tab**.

What we learned:

- What cookies are and when to use them
- Differences between cookies and localStorage
- Use cases for cookies and localStorage

Jason? No, JSON!

Jason is an ancient Greek mythological hero who went traipsing about the known world looking for "the golden fleece".

JSON is an open-standard file format that "uses human-readable text to transmit objects consisting of key-values pairs and array data types."

We're going to ignore Jasonand focus solely on JSON for this reading so that you can, by the end of it, know what JSON is and how to work with it.

JSON is a format!

This is the most important thing that you can get when reading this article. In the same way that HTML is a format for hypertext documents, or DOCX is a format for Microsoft Word documents, JSON is just a format for data. It's just text. It doesn't "run" like JavaScript does. It is just text that contains data that both machines and humans can understand. If you ever hear someone say "a JSON object", then you can rest assured that phrase doesn't make any sense whatsoever.

JSON is just a string. It's just text.

That's so important, here it is, again, but in a fancy quote box.

JSON is just a string. It's just text.

Why all the confusion?

The problem is, JSON *looks* a lot like JavaScript syntax. Heck, it's even named**JavaScript Object Notation**. That's likely because the guy who invented it,Douglas Crockford, is an avid JavaScripter. He's the author of JavaScript: The Good Partsand was the lead JavaScript Architect at Yahoo! back when Yahoo! was a real company.

At that time, like in the late 1990s and early 2000s, there were a whole bunch of competing formats for how computers would send data between one another. The big contender at the time is a format called XML, or the *eXtensible Markup Language*. It looks a lot like HTML, but has far stricter rules than HTML. Douglas didn't like XML because it took a lot of bytes to send the data (and this was a pre-broadband/pre-3G world). Worse, XML is not a friendly format to read if you're human. So, he set out to come up with a new format based on the way JavaScript literals work.

"Remind me about JavaScript literals..."

Just to refresh your memory, a *literal*in JavaScript is a *value that you literally just type in*. If you type `7`into a JavaScript file, when it runs, the JavaScript interpreter will see that character `7` and say to itself, "Hey self, the programmer literally typed the number seven so that must mean they want the value 7."

Here's a table of some literals that you may type into a program.

What you want to type	The JavaScript literal
The value that means "true"	`true`
The number of rows in this table	`6`

What you want to type	The JavaScript literal
A bad approximation of π	`3.14`
An array that contains some US state names	`["Ohio", "Iowa"]`
An object that represents Roberta	`{ person: true, name: "Roberta" }`

Back to Douglas Crockford, inventor of JSON. Douglas thought to himself, *why can't I create a format that has that simplicity so that I can write programs that can send data to each other in that format?*Turns out, he could, and he did.

Boolean, numeric, and null values

The following table shows you what the a JavaScript literal is in the JSON format. Notice that *everything*in the JSON column is actually a string!

JavaScript literal value	JSON representation in a string
`true`	`"true"`
`false`	`"false"`
`12.34`	`"12.34"`
`null`	`"null"`

String literals in JSON

Say you have the following string in JavaScript.

```
'this is "text"'
```

When that gets converted into the JSON format, you will see this:

```
"this is \"text\""
```

First, it's important to notice one thing: JSON always uses double quotes for strings. Yep, that's worth repeating.

JSON always uses double-quotes to mark strings.

Notice also that the quotation marks (") are "escaped". When you write a string surrounded by quotation-marks like "escaped", everything's fine. But, what happens when your string needs to include a quotation mark?

```
// This is a bad string with quotes in it
"Bob said, "Well, this is interesting.""
```

Whatever computer is looking at that string gets really confused because once it reads that first quotation mark it's looking for another quotation mark to show where the string ends. For computers, the above code looks like this to them.

```
"Bob said, "                // That's a good string
Well, this is interesting   // What is THIS JUNK????
""                          // That's a good string
```

You need a way to indicate that the quotation marks around the phrase that Bob says should belong *in*the string, not as a way to show where the string starts or stops. The way that language designers originally addressed this was by saying

If your quotation mark delimited string has a quotation mark in it, put a backslash before the interior quotation mark.

Following that rule, you would correctly write the previous string like this.

```
"Bob said, \"Well, this is interesting.\""
```

Check out all of the so-called JavaScript string escape sequencesover on MDN.

What happens if you had text that spanned more than one line? JSON only allows strings to be on one line, just like old JavaScript did. Let's say you just wrote an American sentence that you want to submit to a contest.

```
She woke him up with
her Ramones ringtone "I Want
to be Sedated"
```

(from American Sentences by Paul E. Nelson)

If you want to format that in a string in JSON format, you have to escape the quotation marks *and* the new lines! The above would look like this:

```
She woke him up with\nher Ramones ringtone \"I Want\nto be Sedated\"
```

The new lines are replaced with "\n".

Array values

The way that JSON represents an array value is using the same literal notation as JavaScript, namely, the square brackets `[]`. With that in mind, can you answer the following question before continuing?

What is the JSON representation of an array containing the numbers one, two, and three?

Well, in JavaScript, you would type `[1, 2, 3]`.

If you were going to type the corresponding JSON-formatted string that contains the representation of the same array, you would type `"[1, 2, 3]"`. Yep, pretty much the same!

Object values

Earlier, you saw that example of an object that represents Roberta as

```
{ person: true, name: "Roberta" }
```

The main difference between objects in JavaScript and JSON is that the keys in JSON *must* be surrounded in quotation marks. That means the above, in a JSON formatted string, would be:

```
"{ \"person\": true, \"name\": \"Roberta\" }"
```

Some terminology

When you have some data and you want to turn it into a string (or some other kind of value like "binary") so your program can send it to another computer, that is the process of **serialization**.

When you take some text (or something another computer has sent to your program) and turn it into data, that is the process of **deserialization**.

Using the built-in JSON object

In modern JavaScript interpreters, there is a `JSON`object that has two methods on it that allows you to convert JSON-formatted strings into JavaScript objects and JavaScript object into JSON-formatted strings. They are:

- `JSON.stringify(value)`will turn the value passed into it into a string.
- `JSON.parse(str)`will turn a JSON-formatted string into a JavaScript object.

So, it shouldn't come as much of a surprise how the following works.

```
const array = [1, 'hello, "world"', 3.14, { id: 17 }];
console.log(JSON.stringify(array));
// prints [1, "hello, \"world\"", 3.14, {"id":17}]
```

It shouldn't surprise you that it works in the opposite direction, too.

```
const str = '[1,"hello, \\"world\\"",3.14,{"id":17}]';
console.log(JSON.parse(str));
// prints an array with the following entries:
//   0: 1
//   1: "hello, \"world\""
//   2: 3.14
//   3: { id: 17 }
```

You may ask yourself, "What's up with that double backslash thing going on in the JSON representation?". It has to do with that escaping thing. When JavaScript reads the string the first time to turn it into a `String`object in memory, it will escape the backslashes. Then, when `JSON.parse`reads it, it will still need backslashes in the string. This is all really confusing, escaped strings and double backslashes. There's an easy solution for that.

You will almost never write raw JSON

Yep. But, you do need to be able to recognize it and read it. What you'll likely end up doing in your coding is creating values and using `JSON.stringify`to create JSON-formatted strings that represent those values. Or, you'll end up calling a data service which will return JSON-formatted content to your code which you will then use `JSON.parse`on to convert the string into a JavaScript object.

Brain teaser

Now that you know JSON is a format for data and is just text, what will the following print?

```
const a = [1, 2, 3, 4, 5];
console.log(a[0]);

const s = JSON.stringify(a);
console.log(s[0]);

const v = JSON.parse(s);
console.log(v[0]);
```

What you just learned

With some more practiced, of course, you will be able to do all of these really well. However, right now, you should be able to

1. Identify and generate valid JSON-formatted strings
2. Use `JSON.parse`to deserialize JSON-formatted strings
3. Use `JSON.stringify`to serialize JavaScript objects
4. Correctly identify the definition of "deserialize"
5. Correctly identify the definition of "serialize"

Using Web Storage To Store Data In The Browser

Like cookies, the Web Storage APIallows browsers to store data in the form of key-value pairs. Web Storage has a much larger storage limit than cookies, making it a useful place to store data on the client side.

In the cookies reading, we reviewed the two main mechanisms of Web Storage:`sessionStorage`and `localStorage`. While `sessionStorage`persists for the duration of the session and ends when a user closes the browser, `localStorage`persists past the current session and has no expiration date.

One typical use case for local storage is caching data fetched from a server on the client side. Instead of making multiple network requests to the server to retrieve data, which takes time and might slow page load, we can fetch the data once and store that data in local storage. Then, our website could read the persisting data stored in localStorage, meaning our website wouldn't have to depend on our server's response - even if the user closes their browser!

In this reading, we'll go over how to store and read a key-value pair in local storage.

Storing data in local storage

Web Storage exists in the window as an object, and we can access it by usingWindow.localStorage. As we previously reviewed, with window properties we can omit the *"window"*part and simply use the property name, `localStorage`.

We can set a key-value pair in local storage with a single line of code. Here are a few examples:

```
localStorage.setItem('eatz', 'I <3 falafel');
localStorage.setItem('coffee', 'black');
localStorage.setItem('doughnuts', '["glazed", "chocolate", "blueberry",
"cream-filled"]');
```

The code above calls the `setItem()`method on the Storage object and sets a key-value pair. Examples: `eatz`(key) and `I <3 falafel`(value), `coffee`(key) and `black`(value), and `doughnut`(key) and `["glazed", "chocolate", "blueberry", "cream-filled"]`(value). Both the key and the value must be strings.

Reading data in local storage

If we wanted to retrieve a key-value pair from local storage, we could use `getItem()`with a key to find the corresponding value. See the example below:

```
localStorage.setItem('eatz', 'I <3 falafel');
localStorage.setItem('coffee', 'black');
localStorage.setItem('doughnuts', '["glazed", "chocolate", "blueberry",
"cream-filled"]');

const eatz = localStorage.getItem('eatz');
const coffee = localStorage.getItem('coffee');
const doughnuts = localStorage.getItem('doughnuts');

console.log(eatz); // 'I <3 falafel'
console.log(coffee); // 'black'
console.log(doughnuts); // '["glazed", "chocolate", "blueberry", "cream-filled"]
```

The above code reads the item with a key of `eatz`, the item with a key of`doughnut`, and the item with a key of `coffee`. We stored these in variables for handy use in any function we write.

Check the MDN docs on localStoragefor other methods on the Storage object to remove and clear all key-value pairs.

JSON and local storage

When we store and read data in local storage, we're actually storing JSONobjects. JSON is text format that is independent from JavaScript but also resembles JavaScript object literal syntax. It's important to note that JSON exists as a *string*.

Websites commonly get JSON back from a server request in the form of a text file with a `.json`extension and a MIME type of `application/json`. We can use JavaScript to parse a JSON response in order to work with it as a regular JavaScript object.

Let's look at the `doughnuts`example from above:

```
localStorage.setItem('doughnuts', '["glazed", "chocolate", "blueberry",
"cream-filled"]');
const doughnuts = localStorage.getItem('doughnuts');
console.log(doughnuts + " is a " + typeof doughnuts);
// prints '["glazed", "chocolate", "blueberry", "cream-filled"] is a string'
```

If we ran the code above in the browser console, we'd see that `doughnuts`is a string value because it's a JSON value. However, we want to be able to store`doughnuts`as an *array*, in order to iterate through it or map it or any other nifty things we can do to arrays.

We can construct a JavaScript value or object from JSON by parsing it:

```
const doughnuts = JSON.parse(localStorage.getItem('doughnuts'));
```

We used JSON.parse()to parse the string into JavaScript. If we printed the parsed value of `doughnuts`to the console, we'd see it's a plain ol' JavaScript array!

See the MDN doc on Working with JSON for more detail about using JSON and JavaScript.

What you learned:

- Why we use local storage
- How to store data in local storage
- How to read data in local storage
- How storage objects are JSON that we need to parse