ITC 210a Tutorial Blog

Jonathan Houston 2/21/2025

Storing, accessing, and displaying JSON data in local storage.

Introduction:

In this tutorial, we will be learning how to use JavaScript to store data from a webpage on a user’s computer locally. This will primarily be using HTML and Javascript, along with a sprinkling of CSS.

HTML, “HyperText Markup Language,” is used to create the bones of a webpage. When you visit a website, your browser receives data from an HTML page (a file with the .html extension) that tells it where the basic parts of a webpage go.

JavaScript is a programming language that can be in an HTML page or included in its own separate file (with a .js extension) and linked in an html page. It runs when called in the HTML page by the user’s browser.

CSS is a programming language that gives the HTML style. By default, HTML will be a rather ugly black and white with boring fonts and simple buttons. CSS allows you to spice that up by adding style to the HTML. Like JavaScript, it can be located inside the HTML file or in a CSS file (with the .css extension) and linked in the HTML file.

JSON is an acronym for “JavaScript Object Notation.” It is used to represent data objects. Most types of data in JavaScript can be converted into a JSON object, and JSON objects can be easily turned into strings. These strings can then be put into local storage.

Local storage allows a site to store key-value pairs. This stores a name and a string. It is very useful, as it allows a website to store information on the user’s computer rather than on a database or server. Local storage can be used to remember visits to a website, save half-filled-out form data, and many other useful functions. Each key-value pair can only be 5 MB. Local storage is unique to an individual web page, and can't be accessed by web pages where it wasn't set.

Tutorial:

Setting Up HTML

First, let’s set up a basic HTML page. Create a file called “index.html” and paste the following code into it.

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
```

```
    <title>My First Local Storage!</title>

  </head>

  <body>
    <h1>My First Local Storage!</h1>

  <script src="script.js"></script>

  </body>

  </html>
```

This sets up a basic HTML page that will display “My First Local Storage!” to the screen and to your browser tab’s title.

List

Next up, we want to create our list. You can do this two different ways, depending on if you want your list to be numbered or to be sequenced with bullet points.

For a numbered list, use an “ordered list” in HTML:

```
<ol>

</ol>
```

For a bullet point list, use an “unordered list”:

```
<ul>

</ul>
```

This tutorial uses the ordered list, but it will work the same for the unordered list. Make sure to put this above the script tags.

HTML Input Forms

Next, we want to create a way for users to add items to the list. We’re going to do this via a form in the HTML. We'll also want a button that will delete all items from the list.

```
<form>
    <input type="text" name="text" required/>
    <br>

    <button>Add Item</button>
    <br>
</form>

<form>
    <button>Delete List</button>
</form>
```

This will create a text box where users can enter the text that they want to add to the list, and a button that will allow them to delete the contents of the list. Neither of these will do anything at the moment. We will make them work with JavaScript, which we will start on next.

JavaScript

To start on the Javascript, create a file called “script.js” in the same folder as the HTML file. This is the file where we will be doing the bulk of our work.

"Item" class

The first thing we want to do is create a class object called “Item”. Classes let you create a custom variable type that has specific properties.

Make sure you note the capital I. In this class, we want to have a constructor (a function that puts the item together) and a function to turn the text into valid HTML.

```
class Item {

    constructor({text}){
        this.text = text;
    }

    toHTML(){
        return "<li>" + this.text + "</li>";
    }

}
```

The constructor will be called whenever a new Item is created. It creates a variable “text” that belongs to the Item, set to the text that was submitted in the constructor. The toHTML function is a function that will turn the text of the task into a valid bit of HTML code that can be displayed on the HTML page.

Definition of the List

Next, we will create an array of Items. An array is a list of variables that are indexed starting from 0. This will be the variable where we will be storing all of the Items, and what we will be storing as a JSON object. Since we don’t want any items to be in the list, we create it like this:

`let list = [];`

If you wanted to start the list with a set task, you could enter something like:

```
let first = new Item({
    "text":"My first item!"
});

list.push(first);
```

This creates a new Item called first, then pushes it to the end of the list array.

Functions

Next, we want to write the functions that will make the list work. First, let’s create the function that saves the list array in local storage.

Store Data in Local Storage

Local storage can only store key-value pairs. That means that it can store a 'string,' or list of characters, which has a specific name.

```
function updateStorage() {
    let data = JSON.stringify(list);
    localStorage.setItem("list", data);
    return;
}
```

Let’s go through the function line by line. First, we create a variable named data by turning list into a JSON string. If we saved it with only the first task, it would be this string:

`[{“text”:”My first item!”}]`

If we added more items, the array would look like this:

`[{“text”:”My first item!”},{“text”:”My second item.”},{“text”:”A third item?”}]`

The entire array would be encompassed by the brackets and each item in the list would be inside the curly braces. On the next line, we set the item “list” in local storage to be data. This saves the JSON string in the local storage under the name “list”. On the final line, we end the function with return, and then have a closing curly brace.

Read Storage into an Array

The next function we will want to write is the function that will get the data out of storage and put into an array.

```
function readStorage(){

    const storage = JSON.parse(localStorage.getItem('list')) || [];
    let result = storage.map(
        listData => new Item(listData)
    )

    return result;
}
```

This function begins by creating a const variable named “storage.” The const part makes it a variable that can’t be changed; this isn’t really necessary, it’s just for convenience. The “JSON.parse” takes the long string and turns it into a generic JSON object–or in this case, an array of JSON objects. If it doesn’t find anything saved to local storage, it sets the storage variable to be an empty array.

The function then creates a new variable called “result.” It uses the .map function on the storage variable, which maps the contents of the array onto other variables. When called like this, it iterates over the entire list in storage, placing each JSON object into an Item object. It declares each individual item in the array as listData, then creates an Item based on listData. This new item is then added to the result array. If the listData was blank, it would do nothing. Once the map function is completed, it has created an array of objects that is identical to the list, and sets the result variable to it. It then returns the result variable.

Reading the List into HTML

Next, we need to make a function to insert the HTML of the list items into the index page. For this, we will need to go back to the index page. We will add the following to the opening tag of the list like so:

`<ol class="list">`

This adds the list CSS class to the HTML list. This can be used in a linked CSS file to change the list’s properties. You could make the text of the list green, or use a particular font, but CSS is not within the scope of this tutorial. Adding the CSS class will allow us to easily target this section of the index page with our JavaScript and insert the proper HTML code for each list object there.

Now, let’s write the function for reading the list into the HTML.

```
function readList(){
    let listElement = document.querySelector('.list');


    listElement.innerHTML = "";

    let storageCheck = localStorage.getItem('list');
    if (!storageCheck){
        return;
    }

    list = readStorage();

   let htmlContent = list.map(item => item.toHTML()).join("")
   listElement.innerHTML = htmlContent;

   event.currentTarget.reset();
```

```
    return;
}
```

First, we create a unique type of variable in “listElement.” The “document.querySelector(.list)” selects all elements with the list class and sets their content to the listElement variable. In the next line, we set the inner HTML of this object to an empty string. Then, we check to make sure there is data saved in the local storage. If there isn’t any, it ends the function. If there is something saved in the local storage, it sets the list variable to it via the readStorage function. Then, we map out the html of each item in the list, join them together without spaces, then set the result to a variable called htmlContent. It’s a long string of HTML code. We then set the inner HTML of the listElement to the htmlContent variable. After that, we reset the form. This clears the text box, meaning that what you typed into it would be erased after being added to the list. This doesn't effect the actual HTML page, but instead the Document Object Model, which is how the browser displays the web page. The DOM is a rather complex topic which you can learn more about in a guide linked in the References and Resources section.

List Deletion

Next, we want to write a function that will delete the list.

```
function deleteList(){
    localStorage.removeItem('list');

    list = [];

    readList();

    return;
}
```

This function deletes the locally stored list item, sets the list variable to an empty array, then rewrites the HTML printed to the list on the index page.

We have the create and delete buttons on the index page, but they don’t add or delete anything. We need to write a pair of functions that will enable the item entry and delete list forms to create new items and delete them.

Linking JS and HTML form

The first function we want to write is as follows:

```
function formSubmit(event){
    event.preventDefault();

    let form = new FormData(event.currentTarget);

    let itemData = Object.fromEntries(form);

    let newItem = new Item({
        text: itemData.text
    })

    list.push(newItem);

    updateStorage();

    readList();

    return;
}
```

This function is designed to be called when a specific event goes off, specifically the submission of the item creation form (which is why there is a variable called event in the parentheses of the function title. The first line of code prevents the default behavior of the event. The function then defines a FormData object called form which is based upon the data submitted in the form.

The contents of form will be defined by the name of the associated element they came from. For this form, that means that the text entered in the bar will be passed to the JavaScript as {“text”:”your text here”} when the user clicks the button. If there were additional elements, they would also be included in the form submission. For example, if there was also a date field in the form, the submission would look like {“text”:”your text here”, “date”:”0000-11-02”}.

A variable called itemData is set to a JSON object created from the data in the form entries. We then create a new Item object called newItem (very creative, I know) from the itemData. The new item is added to the list, and the functions that save the list and then read it out to the HTML are called.

To connect this function with the form submission, we need to return to our index.html file. Inside the opening tag of the form used to submit new items, add the following:

`<form onsubmit="formSubmit(event)">`

This will run the associated function whenever the form is submitted.

Linking JS and HTML Delete Button

We also want to add a similar function to the delete form. The function for that is far simpler than for submitting an item, since all it needs to do is prevent the default behavior and call the deleteList function.

```
function deleteSubmit(event){
    event.preventDefault();
    deleteList();
}
```

Add an on submit trigger for this function to the form wrapped around the delete button like so:

`<form onsubmit="deleteSubmit(event)">`

Conclusion

And now we are done! You now have a working HTML page that interacts with local storage and uses JSON objects. This is a good first step in making a website. If you want to learn more, look up tutorials on preventing cross-site scripting and setting up CSS. Several are linked in the references section.

References and Resources:

https://developer.mozilla.org/en-US/docs/Learn_web_development/ - MDN Web Docs is a high quality source for all things related to web development. It contains a number of useful courses and bits of information.

https://www.theserverside.com/definition/JSON-Javascript-Object-Notation - A highly readable explanation of what JSON is. While it doesn’t discuss specific uses, this article is useful for those who have no prior experience with JSON.

https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage - MDN Web Docs’ article on local storage in JavaScript and what it does. This page is very brief, concise, and to the point.

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API - MDN Web Docs' walkthrough of how to use the web storage API. This goes into more detail on how to use local storage than the above article on local storage.

https://flaviocopes.com/dom/ - An article detailing the Document Object Model, which is only briefly touched on in this tutorial.