LOCAL STORAGE

Back Home (/#/)

INTRO

In this lesson we're going to learn about Local Storage - our first foray into persisting data! Specifically we'll cover:

- What Local Storage is
- Setting items
- Getting items
- Removing items
- Clearing the whole thing
- Storing complex data

No need for more hyping, let's dive right in.

WHAT IS LOCAL STORAGE?

Local Storage is a browser technology that gives web developers the ability to temporarily (perhaps permanently*) store data right in the browser. This does **not** replace databases, but it is used for storing simpler things (like a shopping cart list, user site-preferences, etc.) that we don't need/want to store in a database.

*Theoretically, stuff you save to local storage could stay there forever. You'll understand soon enough =]

There's no hard-and-fast rule for *when* to use Local Storage - it's an architectural decision you'll learn to make as you code more. But there are some cases where you pretty much have to use it (as with authentication, which we are not covering right now).

KEY-VALUE STORE

Ultimately, Local Storage is merely a plain object, stored in your browser, that hold **keys** and **values** - should be familiar from your work with JS/JSON.

Open the Console tab in the Chrome DevTools. All three of the lines below do the *same* thing. Pick one, type it into the console, and then hit enter.

```
// Functions
localStorage.setItem('name', 'Shoobert');

// Object
localStorage.name = 'Shoobert';

// Object
localStorage['name'] = 'Shoobert';
```

Now if you type `localStorage` into your dev console, you should see it as an object with `name` as one of the keys. Likely you'll see other stuff that's been utilizing your local storage as well (if you see `ea-learn-app-token`, that's the coding bootcamp site storing some information about you!)

Refresh the browser. Still see the `name` key? Yep! That's because local storage persists data in the browser itself.

Aside from the ways we'll teach you to remove things from Local Storage (as develoeprs), the only way a user can accidentally (or on purpose) clear their local storage is via these conditions (https://stackoverflow.com/questions/8537112/when-is-localstorage-cleared) - it depends on the browser.

Also important to note, there is no (reasonable) limit to how much we can store in Local Storage. But if you use the same `key` twice, the second time will overwrite the first - as we know.

We'll see soon how we can store things aside from strings.

GETTING ITEMS

Getting items is just as easy as setting them. Remember: `localStorage` is just an object, so there's nothing preventing us from accessing our data as we would from a normal object:

```
localStorage.name
//'Shoobert'
```

But, because `localStorage` is a special object with extra methods, we can also use `getItem`:

```
localStorage.getItem('name')
//'Shoobert'
```

Spot check: it's important to note that `localStorage` is an object that is *built-in* to JS. To that end, **open up some JS file that's connected to an HTML file**. Then:

- In your JS file, save some item to Local Storage
- Open up your HTML in the browser, and check the Console to see that your item is there
- Back in your JS file, write some code that gets the item's value from Local Storage, and `console.log` it

REMOVING

Of course, we also want to remove things from LS every now and again. The following deletes the key `name` (and of course its value):

```
localStorage.removeItem('name')
```

We can also go ham and clear the whole thing:

```
localStorage.clear()
```

STORING COMPLEX DATA

We said that LS works like an elaborate object, but if you try to do something like this (in Chrome, at least):

```
localStorage.personalData = {
  averageTimeOnSite: {unit: "hr", amt: 9},
  probabilityOfFriends: 0.02,
  commonKeywords: ["salsa for one", "1 vs. none Chess"]
}
```

And then you try to extract that using `localStorage.personalData`, you might see this as a result:

```
"[object Object]"
```

Good old `[object Object]`, always showing up where we don't want it. Anyway.

Thankfully, the solve is pretty simple. The expectation is that you store your data as a *string*, no matter what the data is (even an object as complex as the `personalData` key above.)

We can use JS' built-in `JSON` object, and take advantage of its `stringify` method, like so:

```
localStorage.personalData = JSON.stringify({
  averageTimeOnSite: {unit: "hr", amt: 9},
  probabilityOfFriends: 0.02,
  commonKeywords: ["salsa for one", "1 vs. none Chess"]
})
```

Note that we can use `JSON.stringify` anywhere, not just with Local Storage:

```
let x = {name: "Shoobert"}
x = JSON.stringify(x)
console.log(x) //prints a string-version of the object
```

The problem with storing things in LS as a string, though, is that then when we access them we cannot use them as objects. In other words, this won't work:

```
let userStuff = localStorage.personalData //the object we stored earlier

console.log(userStuff.probabilityOfFriends) //prints undefined
```

It doesn't work because `localStorage.personalData` returns a string - and strings don't have a `probabilityOfFriends` property.

However, we *can* extract the object using another popular method of `JSON` called `parse`:

```
let userStuff = JSON.parse(localStorage.personalData)

console.log(userStuff.probabilityOfFriends) //prints 0.02
```

Now our `userStuff` variable is an actual object, and not a string - and that wraps up our LS cycle.

Spot check: in the same JS/HTML files from before, use the data below to:

- Save `userStorage` to LS
- Check that it's all good in the browser - access the count of the last item in `cart` and make sure you get 16

```
let userStorage = {
    darkMode: true,
    showSideNav: false,
    defaultResultCount: 9,
    latestMarks: {
        sectionA: 92,
        sectionB: 11
    },
    cart: [
        { id: 3112, count: 2 },
        { id: 812, count: 16 }
    ]
}
```

Show

PRUDENCE & PRACTICE

For this section, you have two options:

- Add Local Storage functionality to an old project that you've already completed
- Complete the exercise below

Of course, you're welcome to do both ;)

Either way, you should be aware of the following concept: we cannot 100% trust that our data is always in Local Storage. The user *can* delete it, inadverntently or otherwise. **Regardless, Local Storage works per browser**, so it's enough that our user switched computers, and our LS won't be consistent.

As such, when retrieving data from LS, we always want to guarantee that we have a failsafe. Here is a simple example:

```
let data = JSON.parse(localStorage.dataThatIsImportant) || []

data.forEach(d => /*something with data*/)
```

Here our application assumes that `data` is some array. It could be user preferences, or some items in a shopping cart - whatever, but if the user deleted `dataThatIsImportant`, or changed computers, then **`localStorage.dataThatIsImportant` is undefined!**

And of course, we cannot do `undefined.forEach...`

So we use the `something || []` 'trick' to say *give us something,* ***or*** *if that thing is somehow falsy, give us an empty array*.

Of course, you could also do `... || {}`, or any other default you like - but make sure you have a failsafe when working with Local Storage!

Alright, now for **some practice**. Here are you instructions:

- Create a simple input and button
- When the user presses the button, it should display the text they wrote on the screen below the input
  - As well, each piece of text should be added to an array called `wisdom`
  - You should push the text as an object that looks like this: `{text: "the thing the user wrote"}`
- On every other click (i.e, whenever the length of `wisdom` is even), save all of `wisdom` to Local Storage
- When you refresh the page, you should load all the text from `wisdom` and display it under the input right away

Once you're done with that, complete the following:

- Add a "Clear Wisdom" button that, when pressed, removes `wisdom` entirely from Local Storage
- Add a little `x` next to each wisdom (on the HTML), that deletes *that* specific piece of wisdom from Local Storage

  - Might be useful to add an `id` to each piece of wisdom

That's all, you've got this ;)

FINAL REMARKS

Again, LS **does not** replace databases. But still, it comes in useful when we want to store things that we *don't* necessarily want in our database.

Enjoy your Local Storage ~