saving preferences in the user's browser

In working on another idea (how to allow users to select their prefered reading configuration on a web page) I came across another issue: How do we allow users to save their preferences so they don't have to redo their work every time they visit the content.

I remembered an old project that I worked on to test the same hypothesis from a different point of view.

Rather than use that project, I've created the full project in Github. I will only highlight details that I think are important.

The idea is that after the user changes the settings, the browser will save them to local storage and then, when the user returns, it will use the values from local storage as the new values for the font parameters.

I hear the complaint that this will have to be done for each individual browser the user works with... That's true but I'm ok with it, I don't expect every single display to look the same and I don't expect readers to do this on their own unless they need to.

Now that we've set the parameters and objectives, let's look at the code.

The first part is an HTML box with multiple sliders, one for each attribute that we want to change. The example below only handles the weight of the font. I've deliberately choosen to use one decimal place for the value as I'm not certain readers would be able to tell the difference.

```
<div class="settings">
  <fieldset>
    <legend>Font Settings: Roboto</legend>

    <label for="robotoWeight">Weight</label>
    <input  type="range"
            id="robotoWeight"
            name="robotoWeight"
```

```
            min="400" max="900"
            value="400" step="0.1">
    <p><strong>Weight</strong>:
    <span class="weightSlider"></span></p>
  </fieldset>
</div>
```

In the CSS section we take advantage of CSS variables and the fact that they are “live”, if we change the value of a variable it will automatically reflect on the page.

We import the font that we will use in the projoect, Roboto Variable.

We then set up our variables in the `:root` pseudo element. `:root` is similar to the `html` element but it has higher specifity.

The final part of the CSS block is to use the variables using the `var()` function. We’re still using `font-variation-settings` to make sure the code works in as many browsers as possible.

```
@font-face {
  font-family: Roboto;
  src: url("fonts/Roboto-min-VF.woff2");
}

/* Defaults */
:root {
  --line-height: 1;
  --font-weight: "wght" 100;
  --font-width: "wdth" 100;
  font-family: Roboto, sans-serif;
  font-size: 100%;
}

"wght".content {
  line-height: var(--line-height);
  font-variation-settings:
    var(--font-weight),
    var(--font-width);
```

```
}
```

The Javascript block is where the magic happens.

First we define two functions.

The first one tests if we support local storage by creating and removing an item inside and `return true` if the activity suceeds.

If we cannot set or remove an item the code will fail, log a message to console and return false.

We use a try/catch block to ensure that we can return from each branch and that both success and failure will be handled appropriately.

```
function hasLocalStorage() {
  try {
    localStorage.setItem(mod, mod);
    localStorage.removeItem(mod);
    return true;
  } catch (e) {
    console.log('Local Storage Not Supported');
    return false;
  }
}
```

The second function is a convenience function to insert rules into the `:root` pseudo-class in the base stylesheet.

It first captures a reference to the `:root` CSS rule in our stylesheet.

Then we build the CSS variable by setting a property in our stylesheet rule. We add the two dashes (`--`) requuired for CSS variables and the name, with the value as the second parameter.

```
function setRootVar(name, value) {
  let rootStyles = document.styleSheets[0].cssRules[1].style;
```

```
  rootStyles.setProperty('--' + name, value);
}
```

We use the oninput handler to tell the browser what to do when the content of the input element changes.

In this case we call `setRootVar` to set the font-weight CSS variable using the string "wdth" and the value of the slider as the second parameter. I decided to go the extra mile so it would be easier to build the variable and use it when we update the `font-variation-settings` CSS.

I've also stored two elements in local storage:

One is the full value of `font-weight`: the string and the value of the slider.

The other one is just the value of the weight slider. I've done this to make it easier on myself when retreiving the data later.

```
weight.oninput = function() {
  weightSlider.innerHTML = weight.value;
  // setting the style
  setRootVar('font-weight', ' "wght" ' + weight.value);
  localStorage.setItem('font-weight', ' "wght" ' +'font-weight');
  localStorage.setItem('weight-value', weight.value);
};
```

The last block is a DOMContentLoaded event handler to retrieve the settings from localStorage, set the font attributes accordingly and provide defaults if the attribute is not stored in local storage or is empty.

If the attribute exists and is not empty then we update the position of the slider and the value it reflects. This way the user will not have to redo the sliders with the values they wanted (and that are reflected on the text).

If the value is not set or is null, we provide defaults that match the values we set in the CSS stylesheet.

```
window.addEventListener('DOMContentLoaded', event => {
```

```
  if (localStorage.getItem('font-weight') &&
    localStorage.getItem('font-weight') !== null) {
    setRootVar('font-weight', localStorage.getItem('font-weight'));
    robotoWeight.setAttribute(
      'value',
      localStorage.getItem('weight-value'),
    );
    weightSlider.innerHTML = localStorage.getItem('weight-value');
  } else {
    setRootVar('font-weight', 400);
    robotoWeight.setAttribute(
      'value',
      localStorage.getItem(400),
    );
    weightSlider.innerHTML = 400;
  }
});
```

There are some things I'm still working on. Some times the values do not load properly and I'm trying to figure out why.

This is a first step in providing a way to save settings for an app. We might want to expand the test to something closer to a full-blown reading application.