Assignment 4: Temperature Conversion Page

LIB2024 Internet Applications

By Martha Kohler
040864508
April 17, 2023

Contents

Introduction

The purpose of this assignment was to build a simple webpage using HTML, CSS, and JavaScript with the function of calculating temperature conversions from Celsius to Fahrenheit. Another component was to code an error response if a non0-numeric field was entered into the input field.

Link to Github

The assignment folder includes HTML, CSS, and JavaScript files as well as assets, this report and a screen video of the page's function here:

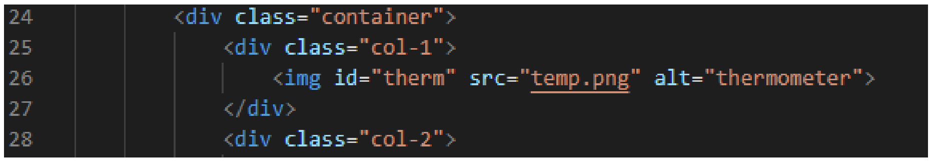
https://github.com/kohl0028/kohl0028.github.io/tree/main/lib2024/Assignment%204%20Martha%20Kohler

Coding

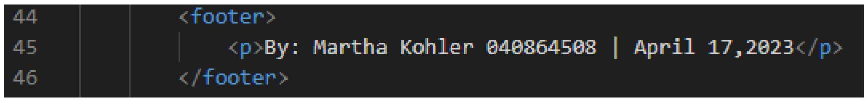
HTML

i. <header> &<footer>

The <header> tag was used to place the title at the top of the page. The <h1> tag was chosen to represent the importance of the content within. Similarly, a <footer> was added to contain publishing information and signal the end of the page content.



```
    <div class="container">
      <div class="col-1">
        <img id="therm" src="temp.png" alt="thermometer">
      </div>
      <div class="col-2">
```



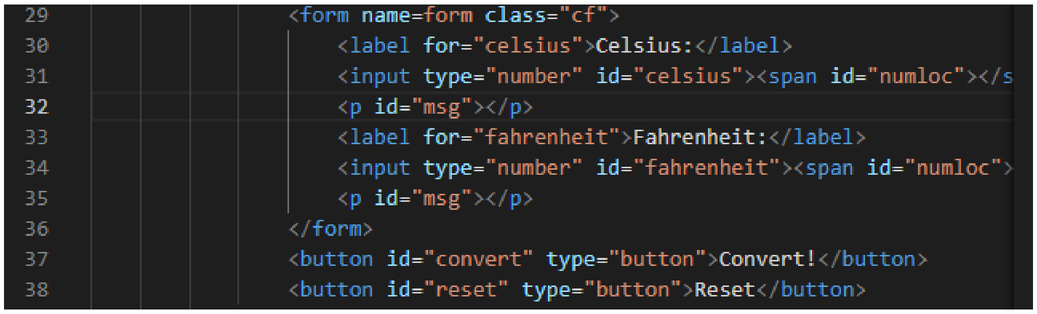
```
      <footer>
        <p>By: Martha Kohler 040864508 | April 17,2023</p>
      </footer>
```

ii. <body, <div>

To structure the main content, <div> one large container was used to hold 2 child <div> buckets. This way, 2 columns of content could be easily styled later. The image file was added to <div class="col-1"> and the input fields and buttons were placed in the second <div class="col-2">.

iii. <form>, <label>, <input>, <button>

Within <div class="col-2"> , the <form> tag was chosen to contain the input fields and buttons. Within this form, there are input fields and labels. The <input> tag tells the server what sort of data will be submitted and can also include more attributes to define the content, like type and id. Because we needed a number for temperature conversion, type="number" was defined. The <label> tells the user what the value to be inputted into the following fields will be and allows the creator to add content. For this assignment, two input fields and corresponding labels were created for Celsius and Fahrenheit inputs. Outside of the <form>, two buttons were added: Convert! and Reset. These buttons were given attributes to allow for styling and to allow for JavaScript functions.

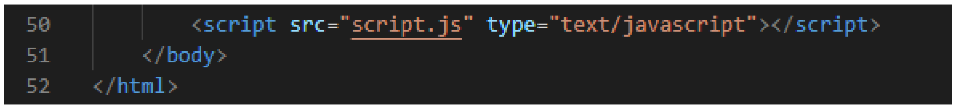


```
<form name=form class="cf">
    <label for="celsius">Celsius:</label>
    <input type="number" id="celsius"><span id="numloc"></s
    <p id="msg"></p>
    <label for="fahrenheit">Fahrenheit:</label>
    <input type="number" id="fahrenheit"><span id="numloc">
    <p id="msg"></p>
</form>
<button id="convert" type="button">Convert!</button>
<button id="reset" type="button">Reset</button>
```

The layout and structure of the buttons and field was heavily influenced and informed by a YouTube video by Future Coders (2021).

iv. <script>

To link the external JavaScript file, the <script> element was linked at the end of the document. The script is placed at the end so as not to interfere with how the browser reads the source code and renders the page. Placing Javacript at the end ensures that functions are applied and run only after the page content and styling has been loaded.

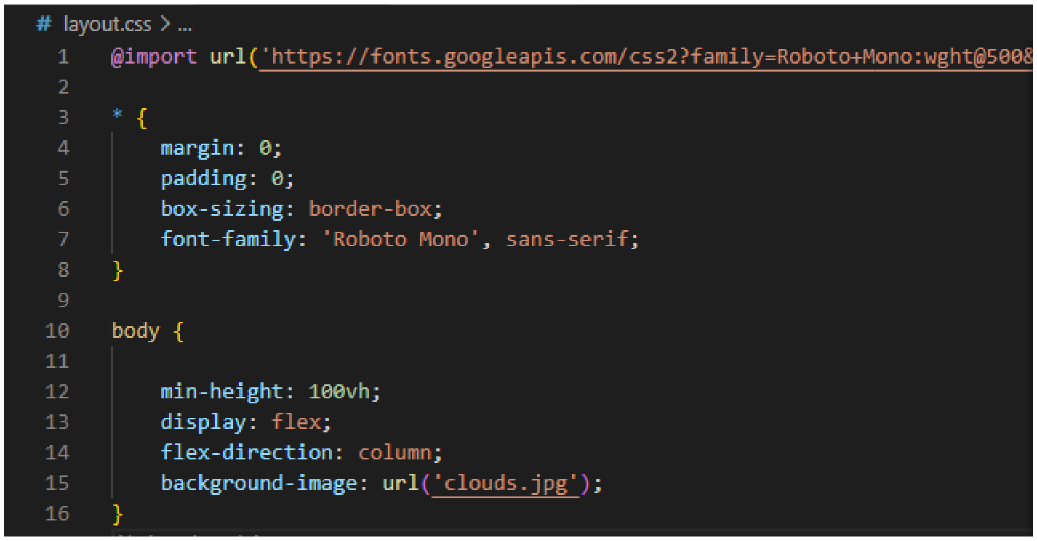


```
        <script src="script.js" type="text/javascript"></script>
    </body>
</html>
```

CSS

i. Background Styling

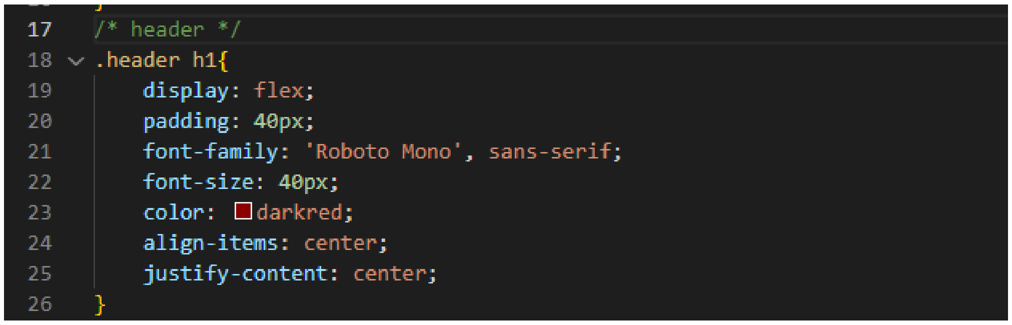
The styling of this simple page is straightforward. Having organized the content into a few <div> buckets in the HTML file, it allowed me to style the elements separately. First, a Google Font was linked using @import, allowing me to reference it by name later in the document. The basic styling aspects for the whole page, like display, margins, padding, and background were applied to the page and body as a whole in lines 3-15 of the CSS. Display: flex was chosen to allow the page contents to have floating positions in relation to each other allow the layout to be responsive to different window sizes.



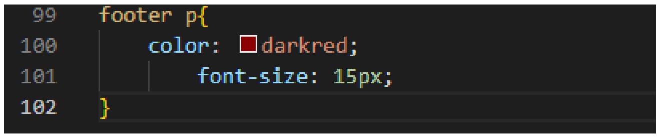
```
# layout.css > ...
@import url('https://fonts.googleapis.com/css2?family=Roboto+Mono:wght@500&

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Roboto Mono', sans-serif;
}

body {

    min-height: 100vh;
    display: flex;
    flex-direction: column;
    background-image: url('clouds.jpg');
}
```

ii. Header & Footer

In the header and footers, the font, alignment, and color were specified. Padding was added to push the content away from the top and sides of the viewport and create visual space.



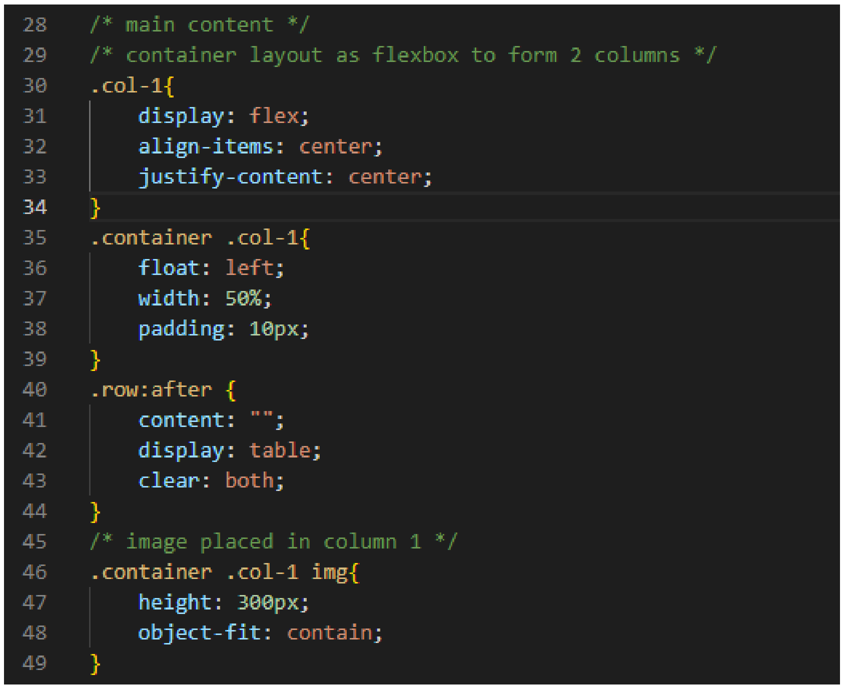
```
/* header */
.header h1{
    display: flex;
    padding: 40px;
    font-family: 'Roboto Mono', sans-serif;
    font-size: 40px;
    color: darkred;
    align-items: center;
    justify-content: center;
}
```



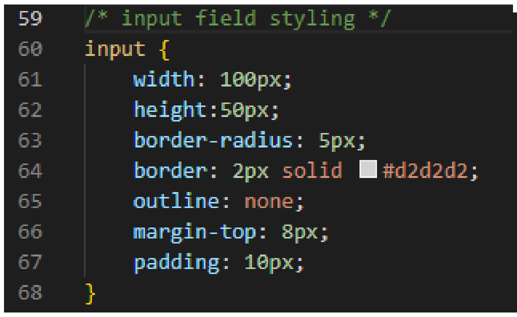
```
footer p{
    color: darkred;
        font-size: 15px;
}
```

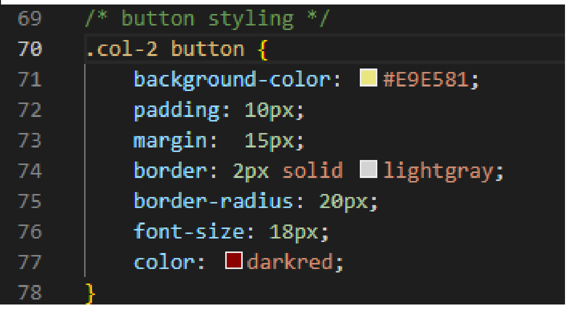
iii. Body

For <body> content, column 1 was assigned a flex display at lines 30-34 Float: left and width: 50% were applied to column 1 to ensure it takes up half the page and stays left. Align-items:center and justify-contents: center ensured the content stays centered within the <div>. The thermometer image was resized and scaled down in lines 46-49.


```css
/* main content */
/* container layout as flexbox to form 2 columns */
.col-1{
    display: flex;
    align-items: center;
    justify-content: center;
}
.container .col-1{
    float: left;
    width: 50%;
    padding: 10px;
}
.row:after {
    content: "";
    display: table;
    clear: both;
}
/* image placed in column 1 */
.container .col-1 img{
    height: 300px;
    object-fit: contain;
}
```

For column 2, which contains the input form and buttons, the containing element was first styled then the input field styling was accomplished by targeting the whole element. The field size, borders, and padding were established, and a border-radius assigned to round the corners slightly (lines 60-68).

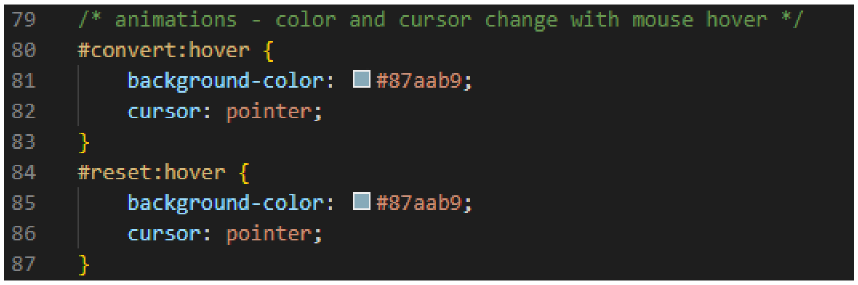

```css
59  /* input field styling */
60  input {
61      width: 100px;
62      height:50px;
63      border-radius: 5px;
64      border: 2px solid #d2d2d2;
65      outline: none;
66      margin-top: 8px;
67      padding: 10px;
68  }
```


```css
69  /* button styling */
70  .col-2 button {
71      background-color: #E9E581;
72      padding: 10px;
73      margin:  15px;
74      border: 2px solid lightgray;
75      border-radius: 20px;
76      font-size: 18px;
77      color: darkred;
78  }
```

The button appearance was defined next, assigning background, font and border colors and sizing (above, lines 70-78).

iv. Animations

Some animations were added to add a little pizzaz to the buttons. For both, the colors were set to change when the mouse hovers over, and the cursor changed to the hand-pointer to indicate a clickable button.

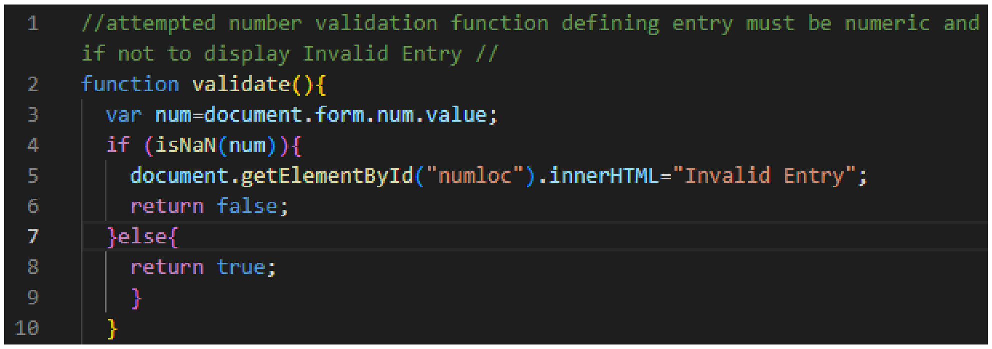


```css
/* animations - color and cursor change with mouse hover */
#convert:hover {
    background-color: #87aab9;
    cursor: pointer;
}
#reset:hover {
    background-color: #87aab9;
    cursor: pointer;
}
```

JavaScript

i. Input Validation

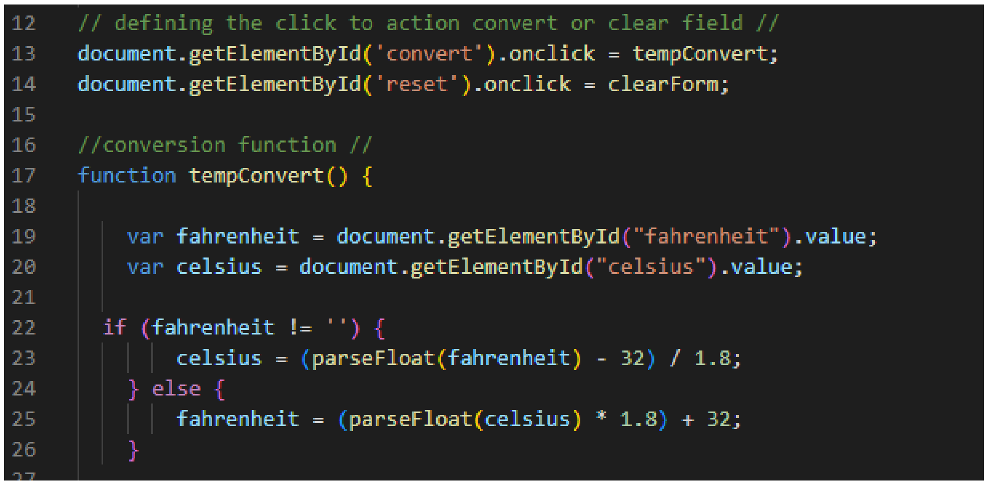
To ensure that the field entry was a numerical value as temperature demands, a validate function was applied to the field using if/else syntax and defining the variable (num). I could not get the function to operate however, and I wonder if my conversion functions are overriding it. The function should offer up a text warning if nonnumeric characters were added, in the <span "id=numloc"> added after the input fields in the HTML.



```javascript
//attempted number validation function defining entry must be numeric and if not to display Invalid Entry //
function validate(){
  var num=document.form.num.value;
  if (isNaN(num)){
    document.getElementById("numloc").innerHTML="Invalid Entry";
    return false;
  }else{
    return true;
    }
  }
```

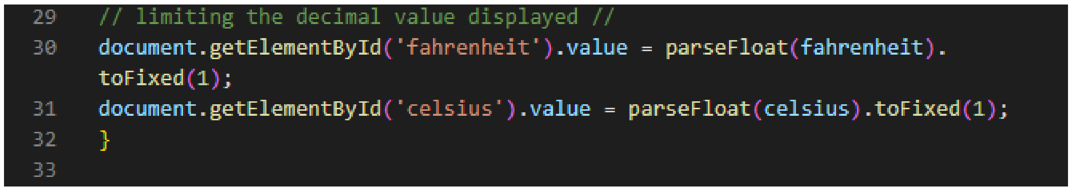
ii. Convert! Button

This functions to convert the numerical input and post a result. The function works both ways, and inputs in either field will generate the corresponding temperature, Celsius or Fahrenheit. First, the targeted elements were identified and called by getElementbyID(). The click action of those elements was defined, with each function to follow. For tempConvert, the if/else syntax was used to apply a mathematical equation to calculate the corresponding temperature fields.



```javascript
// defining the click to action convert or clear field //
document.getElementById('convert').onclick = tempConvert;
document.getElementById('reset').onclick = clearForm;

//conversion function //
function tempConvert() {

    var fahrenheit = document.getElementById("fahrenheit").value;
    var celsius = document.getElementById("celsius").value;

  if (fahrenheit != '') {
      celsius = (parseFloat(fahrenheit) - 32) / 1.8;
    } else {
      fahrenheit = (parseFloat(celsius) * 1.8) + 32;
    }
```

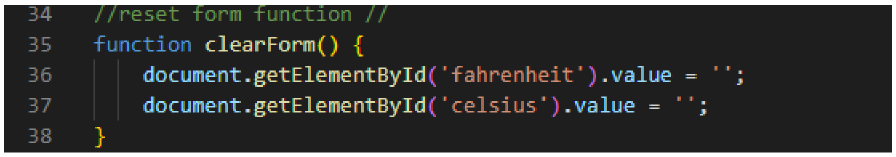
In addition, a function to limit the amount of decimal places in the displayed answer was added:



```
// limiting the decimal value displayed //
document.getElementById('fahrenheit').value = parseFloat(fahrenheit).
toFixed(1);
document.getElementById('celsius').value = parseFloat(celsius).toFixed(1);
}

```

iii. Reset Button

To reset the form fields the function clearForm applies a value of nil on the click, so the input field is rendered blank/reset.



```
//reset form function //
function clearForm() {
    document.getElementById('fahrenheit').value = '';
    document.getElementById('celsius').value = '';
}
```

SEO & UX standards

As always, the <head> contains metadata to ensure effective search engine optimization. For this simple page, there is a description, keywords, and author identified.

For better user experience, the focus was on usability most of all. The title makes clear what the page is about, and there is minimal content to distract the user from the page's purpose. The entry fields are large and the text and form fields centered on the page . There is good contrast in the colors selected, however they aren't too bright. The thermometer image shows the used visually what the page is for as well. Finally, the use of flexbox ensures that the page elements reorganize/adjust depending on viewport size, even with a small window the elements will shrink down and stack and minimize the need for horizontal scrolling to access all the content.

Conclusion

While I was successfully able to code the conversion calculation, I was unable to trigger the error message if a non-numerical value was inputted. If a letter or symbol is entered and the "Convert!" button is clicked, no action is taken, and the input field returns to blank. Ultimately, I was not able to troubleshoot why the function doesn't work as intended. I remain satisfied with the functioning of the page in terms of conversion and clearing the fields.

References

Caparica [username]. (n.d.) *Temperature Converter*. Codepen. https://codepen.io/Caparico/pen/AXOBBY

Fjord, Evie. (2021, January 17). *White clouds and blue sky during daytime* [image]. Unsplash. https://unsplash.com/photos/grCXm-C-eXA

Future Coders. (2021, June 21). *How to Build Temperature Converter with HTML CSS JavaScript | Celsius to Fahrenheit* [Video]. Youtube. https://www.youtube.com/watch?v=MOxPci2QrCE&ab_channel=FutureCoders

JavaTpoint. (n.d.). *JavaScript Form Validation.* https://www.javatpoint.com/javascript-form-validation

Pngfind. (n.d.) *Thermometer clipart* [image]. https://www.pngfind.com/mpng/ohhhwh_thermometer-clipart-thermometer-clipart-transparent-hd-png-download/

Robertson, Christian. (n.d.). *Roboto Mono* [font family]. Google Fonts. https://fonts.google.com/specimen/Roboto+Mono

W3 Schools. (n.d.) *HTML Forms*. https://www.w3schools.com/html/html_forms.asp

W3 Schools. (n.d.). *JavaScript Form Validation*. https://www.w3schools.com/js/js_validation.asp