BTI-225 Assignment 4

Submission Deadline:

Friday, April 7, 2017 @ 11:59 PM

Assessment Weight:

5% of final course grade.

Objective:

Practice Styling elements with CSS, Updating the DOM with data & responding to user events.

Specification (Part A):

For Part A, we will be creating a small demo app that works with "Pet" data, ie, dogs, cats & birds. This app will show all of the pets from a "petData" array of objects and allow the user to filter which pets they would like to see. This app must also be visually appealing, so we must update/override existing CSS to create a more customized experience for the user.

To begin, download the assignment4.zip file containing all of the files required for Assignment 4

assignment4.zip

Uncompress the assignment4.zip file somewhere on your local machine. When you're ready to begin editing the files, open the uncompressed folder (assignment4) and locate the directory "partA". Open this directory using Visual Studio Code (https://code.visualstudio.com). You may test your html files in any modern browser (Chrome, Firefox, Safari, Internet Explorer, etc).:

Updating File: "main.html"

The main page of our app will be the "main.html" file. There is already some code there, but must configure it further to create some interactivity and personalization for the app

1. Give your app a name by updating the <title></title> tag, ie "~ Purfect Pets ~", or whatever you choose
2. Update the <h1 class="app-title"></h1> element to match the name of your app. It currently states "App Title", but this must be changed to something more appropriate (ie. "~ Purfect Pets ~")
3. Update the "onclick" values **all the <a> elements** inside the top <ul> element to call appropriate filtering functions (to be written later). For example clicking "Dogs" should call a function that updates the table to **only show Dogs** (ie, invokes the filterDogs() function – see below)
4. Update the <footer> element to show your full name.

Updating File: "a4-main.js"

All of the JavaScript for the solution (except for the data) is located in the file "a4-main.js". Inside this file, write a series of functions to render the data ("petData" located within js/data.js). You **must not use innerHTML** unless explicitly instructed to do so (there are two instances where it is necessary). The functions are follows:

1. Function: **loadTableWithFilters**

   This function will take all of the data from the global "petData" array (located within js/data.js) and render it as rows within the <tbody id="main-table-body"></tbody> element in the main.html file.

   It must use the global filter values, ie: **filterType** (possible values: "cat", "dog", "bird", or ""), **filterAgeMin** (any positive number) and **filterAgeMax** (any positive number) to conditionally show rows depending on a pet's **type**, or **age** values. For example, if the value of **filterType** is "cat", the value of **filterAgeMin** is **1** and the value of **filterAgeMax** is **3** then only pets with **type: "cat"** who are between the age of **1** and **3** (exclusively) will be added as rows within the <tbody id="main-table-body"></tbody> element.

   Before any rows are added to the <tbody id="main-table-body"></tbody> element however, it must be cleared of all existing data. **This can be done using element.innerHTML = "".**

   Lastly, this function must be invoked inside window.onload to ensure that the <tbody id="main-table-body"></tbody> element is initially populated with all of the pets inside the "petData" array.

   As an example of how each new row must be rendered, consider the following example for **Bella** the dog:

   Data:

```
image: {src: ""https://github.com/allanrandall/BTI225W17/raw/master/Bella.jpg",
alt:"Bella", width:120, height:160},
name: "Bella",
age: 0.5,
description: "<span>Bella</span> is a bright young pup who loves being around other
dogs and doesn't mind the odd cat.<br />Her <span>favourite treat</span> is
<span>bacon</span> - anything <span>bacon</span>.",
type: "dog"
```

   Rendered HTML (<tr></tr> element):

```
<tr>
    <td>
        <img src="https://github.com/allanrandall/BTI225W17/raw/master/Bella.jpg"
alt="Bella" height="160" width="120">
    </td>
    <td>
        <h4>Bella</h4>
        <p><span>Bella</span> is a bright young pup who loves being around other dogs
and doesn't mind the odd cat.<br />Her <span>favourite treat</span> is <span>bacon</
span> - anything <span>bacon</span>.</p>
        <span>Age: 0.5 years old.</span>
    </td>
</tr>
```

   NOTE: **description** for all pets contains **HTML** code; to render this in , **we must use innerHTML**
   (ie: p.innerHTML = petData[i].description)

2. Function: **filterDog**
   This function simply sets the global **filterType** variable to "**dog**" and invokes the **loadTableWithFilters** function again to refresh the table. This function **must be invoked** when the user **clicks "Dogs"**

3. Function: **filterCat**

   This function simply sets the global **filterType** variable to "**cat**" and invokes the **loadTableWithFilters** function again to refresh the table. This function **must be invoked** when the user **clicks "Cats"**

4. Function: **filterBird**

   This function simply sets the global **filterType** variable to "**bird**" and invokes the **loadTableWithFilters** function again to refresh the table. This function **must be invoked** when the user **clicks "Birds"**

5. Function: **filter_zero_1**

   This function simply sets the global **filterAgeMin** variable to **0**, the **filterAgeMax** variable to **1** and invokes the **loadTableWithFilters** function again to refresh the table. This function **must be invoked** when the user **clicks "< 1 year"**

6. Function: **filter_1_3**

   This function simply sets the global **filterAgeMin** variable to **1**, the **filterAgeMax** variable to **3** and invokes the **loadTableWithFilters** function again to refresh the table. This function **must be invoked** when the user **clicks "1 - 3 years"**

7. Function: **filter_4_plus**

   This function simply sets the global **filterAgeMin** variable to **4**, the **filterAgeMax** variable to **Number.MAX_VALUE** and invokes the **loadTableWithFilters** function again to refresh the table. This function **must be invoked** when the user **clicks "4+ years"**

8. Function: **filterAllPets**

   This function simply sets the global **filterType** variable to **""**, the **filterAgeMin** variable to **0**, the **filterAgeMax** variable to **Number.MAX_VALUE** and invokes the **loadTableWithFilters** function again to refresh the table. This function **must be invoked** when the user **clicks "All Pets"**

Updating File: "a4-main.css" – see: completed sample

Now that the table is getting populated with our data and the filters are up and running, we need to make the app a little nicer to look at. This next step involves updating the CSS file "a4-css.css" to clean up the layout and add some colour and style:

1. Add a new font colour to the <body> element
2. Add a new background colour to the <nav> and <footer> elements
3. Style the <header></header> element using at least **2** properties
4. Style the <h1 class="app-title"></h1> using at least **3** properties. One of these properties **must** be setting the font to a "web-font", available on the Google Fonts CDN
5. Style the using at least **3 properties**, two of which must be to ensure that it has:

   - a maximum height value
   - a vertical scroll bar

   This will allow our table to scroll, instead of taking up a ton of vertical space on the page.

6. Add at least **4** style properties to the <th></th> elements inside the <table class="main-table-top"></table> element. This will ensure that the top (stationary) row of our table looks distinctive.
7. Add at least **4** style properties to the <td></td> elements inside the <table class="main-table"></table> element. This will ensure that the cells within our main table have appropriate spacing and are styled to match the rest of the page.
8. Add at least **2** properties to all elements that are used for the description of the pets. These elements will be located inside elements within the <table class="main-table"></table> element. This will ensure that anytime a element is used in the description of a pet, the text will be highlighted.
9. Lastly, update the academic honesty statement at the top of the file with your name and the current date.

Specification (Part B):

For Part B, we will be creating and a new CSS file to be used in the CanadianPT.html file located within the PartB folder.

When you're ready to begin editing the files, open the uncompressed folder (assignment4) and locate the directory "partB". Open this directory using Visual Studio Code (https://code.visualstudio.com). You may test your html files in any modern browser (Chrome, Firefox, Safari, Internet Explorer, etc).

Please note: when developing your CSS, you **must add comments (/* … */)** to describe each of the selectors. For example, above the selector "**.main-container nav**", include the comment **/* Select all <nav> elements that are a child of an element with class "main-container" and adjust it's size and position */**.

Getting Started– completed sample: 1.png

- locate the "**css**" directory and add the file "**site.css**"
- Open the file "**canadianPT.html**" for editing and add the **<link>** tag to include your new **site.css** file
- Open **canadianPT.html** in a web browser: it should look like this: **1.png**
- Open both "**site.css**" and "**canadianPT.html**" for editing/viewing

Step 1 – Styling the main containers – completed sample: 2.png

- Notice how the page "**canadianPT.html**" is broken up into major sections: ** … **, **<header>…</header>, <nav>…</nav>, <section class="main">…</section>** and **<footer>…</footer>**
- We need to apply styles to these sections, so add the following CSS Selectors (in the file **site.css**) to your page. For each selector, use the suggested CSS Properties to achieve each design requirement. See the completed sample for reference

CSS Selector	Design Requirement	Suggested CSS Properties
.main-container	• Set the maximum width to 960 pixels • No top or bottom margins • Left and right margins must be "auto"	• max-width • margin

.main-container nav	• Set the width to 30% • Element is positioned on the left side of the page	• width • float
.main-container .main	• Set the width to 70% • Element is Positioned on the right side of the page	• width • float
.main-container header	• Padding around the content must be 10 pixels • Background colour must be #eeeeee • There must be a solid bottom border 1 pixel thick, coloured #cccccc • Bottom margin must be 30 pixels	• padding • background-color • border-bottom • margin-bottom
.main-container header h1	• Set the width to 70% • The size of the font must be 24 pixels • Element is Positioned on the right side • The text is right-aligned • Top margin must be 10 pixels	• width • font-size • float • text-align • margin-top
.main-container footer	• position the element beneath all floating elements ("clear" the element) • Padding around the top and bottom must be 20 pixels • Padding around the left and right must be 10 pixels • Background colour must be #eeeeee • There must be a solid top border 1 pixel thick, coloured #cccccc • The size of the font must be 13 pixels	• clear • padding • background-color • border-top • font-size

Step 2 – Styling the navigation and logo – completed sample: 3.png

- Now that the page is looking a little more organized, we want to update the "Canada" logo (flag & name) as well as the side navigation to look a little cleaner. For each selector, use the suggested CSS Properties to achieve each design requirement. See the completed sample for reference

CSS Selector	Design Requirement	Suggested CSS Properties

#logo	• Set the width to 30% • The size of the font must be 25 pixels • The font must be bold	• width • font-size • font-weight
#logo img	• The image must be 94 pixels wide • The image must be vertically positioned in the middle, relative to the "Canada" text	• width • vertical-align
nav ul	• Do not show any bullets or numbers for the list	• list-style-type
nav ul a	• render the link as a "block-level" element • remove the underline • Padding around the link must be 5 pixels	• display • text-decoration • padding

Step 3 – Styling the province containers – completed sample: 4.png & 4-resized.png

- Everything is looking pretty good, but the page is very vertical. For a more modern design, we will render all of the provinces as "cards" and allow them to sit side-by-side horizontally. We also want to allow them to move onto the next line and render correctly if the user decides to resize the browser window. For each selector, use the suggested CSS Properties to achieve each design requirement. See the completed samples for reference

CSS Selector	Design Requirement	Suggested CSS Properties
div.province	• Set the width to 30% • Each new element is positioned to the right of the previous element, horizontally (**hint**: float:**left**) • The margin around the element must be 8 pixels • The height of the element must be 310 pixels	• width • float • margin • height

div.province .description	• The element must not be any higher than 165 pixels • Only If the content goes beyond 165 pixels high, show a scroll bar • If the content does not fit in the container horizontally, hide the scrollbar • The top and bottom margins must be 8 pixels	• max-height • overflow-y • overflow-x • margin
div.province a	• render the link as a "block-level" element • position the text in the middle of the element (horizontally) • set the colour of the background to #555555 • Set the colour of the text to #eeeeee • remove the underline • Padding around the link must be 5 pixels	• display • text-align • background-color • color • text-decoration • padding
img.flag	• set the width to be 100% of the parent	• width

Other Requirements

- Each HTML page in Part A & B **MUST PASS (no errors)** the **W3C's HTML Validation**: https://validator.w3.org/
- All CSS used in the assignment **MUST PASS (no errors)** the **W3C CSS Validation**: https://jigsaw.w3.org/css-validator

Assignment Submission:

- Zip **all of your files** (ie, your **assignment4** folder) as **assignment4.zip**
- Upload the zip file to **My.Seneca** under **Assignments -> Assignment 4** (same submission procedure as Assignments 2 & 3)
- **NOTE**: Your **HTML must not contain any errors** when validated (https://validator.w3.org/)
- **NOTE:** Your **CSS must not contain any errors** when validated (https://jigsaw.w3.org/css-validator)