

# SIT 120 Portfolio

Brandon Murch - 221365233

July 18, 2021

# Contents

<b>1</b>	<b>Justification For Grade</b>	<b>2</b>
<b>2</b>	<b>Week 1</b>	<b>3</b>
2.1	Weekly Content . . . . .	3
2.1.1	HTML . . . . .	3
2.1.2	CSS . . . . .	6
2.1.3	JavaScript . . . . .	7
2.1.4	Vue.js . . . . .	8
2.2	Practical Tasks . . . . .	9
2.2.1	1 - HTML . . . . .	9
2.2.2	2 - CSS . . . . .	10
2.2.3	3 - JavaScript . . . . .	10
2.2.4	4 - Vue.js . . . . .	11
2.3	Project . . . . .	12
2.3.1	What I completed this week . . . . .	12
2.3.2	What I will complete next week . . . . .	12
<b>3</b>	<b>Useful Code Snippets</b>	<b>13</b>
3.1	CSS . . . . .	13
3.1.1	Common Properties . . . . .	13
3.1.2	Center an Element . . . . .	13
3.2	JavaScript . . . . .	14
3.2.1	Window.onLoad . . . . .	14
3.2.2	Round to Specific Decimal Place . . . . .	14

# Chapter 1

## Justification For Grade

I will be aiming for a **High Distinction** grade within this assignment, and also more generally throughout the course. I have done a few MOOC courses relating to web development, and have used similar frameworks to VUE, such as React and Angular. Since I am familiar with these topics, I will use this course to fill in the gaps of knowledge, and focus on the more advanced topics within the course, as well as the more formal proposal process.

I will continue to update this justification as the course progresses.

# Chapter 2

## Week 1

### 2.1 Weekly Content

#### 2.1.1 HTML

##### Introduction to HTML

Hyper Text Markup Language is a structured way to store the information which will be displayed on a webpage. More simply, HTML tells the browser **WHAT** to display.

HTML uses tags (example: `< body >< /body >`), with most "opening tags" having a matching "closing tag" with the relevant information between the two. Attributes can be set within the opening tag, to assign classes, ids, etc. to the tag. This looks like: `<p class="text">`

##### Boilerplate To Get Started

There is a small amount of code that is present in most HTML webpages to get started:

---

```
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
    <script src="INSERT_LINK_TO_JAVASCRIPT_FILE_HERE"></script>
    <link rel="stylesheet" type="text/css" href="INSERT_LINK_TO_CSS_FILE_HERE">
  </head>
  <body>
    INSERT HTML TO DISPLAY HERE.
  </body>
</html>
```

---

**Notice the indentation!** To make this code easier to read, any tag that is inside another tag will be indented further. For example, it is easy to see that the head tag is inside the html tag.

<code>&lt;!DOCTYPE&gt;</code>	This is to let the browser know that this is a current HTML5 document. Previous versions of html will have different codes to insert here. (WHATWG 2021)
<code>&lt;html&gt;...&lt;/html&gt;</code>	This is the html document, all information will be inside this tag
<code>&lt;head&gt;...&lt;/head&gt;</code>	This contains information that will not be displayed within the webpage itself.
<code>&lt;title&gt;...&lt;/title&gt;</code>	The title to the webpage. This will be displayed either in the browser's title bar area at the top of the window, or the tab area.
<code>&lt;script&gt;...&lt;/script&gt;</code>	This can either contain JavaScript code directly, or have a src attribute that links to an external file (either locally, or from a http link)
<code>&lt;link&gt;</code>	This links to some exterior document. The most common use of the link tag is to link to an external css file, but other documents that might be linked are icons, and licenses. (W3 Schools n.d.)
<code>&lt;body&gt;</code>	Body contains all the information that will be displayed on the website itself.

### Other useful tags and descriptions

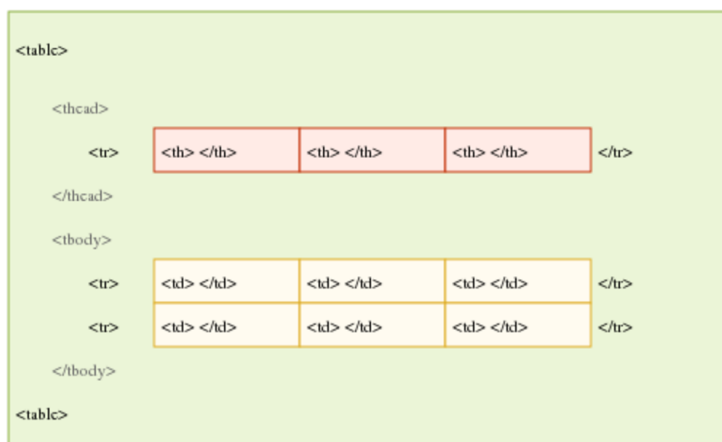
- `<p>...</p>` contains a paragraph of text.
- `<h#>...</h#>` contains a header. # is replaced by the size of header, 1 is the largest and 6 is the smallest.
- `<div>...</div>` is a generic tag that creates a container for other elements.
- `<a href="...">...</a>` creates a hyperlink to another page. href is the link, and the text between the tags is displayed to the user.

### Tags without closing tags

- `` loads an image from either a website, or local storage.

### Tables

This graphic provides a clear understanding of the structure of a table. The information is placed in the inner most tags (th and td).



(HTML/Tabellen/Aufbau einer Tabelle n.d.)

## Forms

Forms are a collection of input fields within HTML. The user can enter different types of information depending on the type of input field. When the form is submitted, the browser will collect all the inputs and either send them to a url (if the attribute **action** is present), or will send them to a javascript function (if the attribute **onsubmit** is present). An example of a form with the inputs name, age and location might be:

---

```
<form onsubmit="submitForm(event)">
  <label for="name">Name</label>
  <input type="text" id="name" size=20 maxlength=20>
  <label for="age">Age</label>
  <input type="number" id="age" max=120>
  <label for="location">Location</label>
  <input type="text" id="location">
  <input class="button" type="submit" value="Submit">
  <input class="button" type="reset" value="Cancel">
</form>
```

---

Which would output:

Name	<input type="text"/>
Age	<input type="text"/>
Location	<input type="text"/>

## Inputs

In HTML5, inputs can have a variety of types (To name a few: text, number, email). Some inputs limit what can be inputted into the field, such as an input with the type email will only accept properly formatted emails. Another reason to use proper input types is to create a responsive website, where a different keyboard will appear on a mobile device depending on the input type.

Your comment:

Your email:

Please enter an email address.

(Mozilla n.d.[d])

Input fields can also be displayed as buttons, for example if the input type is submit or reset as seen in 2.1.1.

For accessibility reasons, each input must be accompanied by a label tag, which uses the **for** attribute to connect to an input's id.

## 2.1.2 CSS

### Introduction to CSS

Cascading Style Sheets tells the browser **HOW** to display the information in the HTML file.

### Basic Syntax

CSS uses property name and value pairs separated by a colon, and different pairs are separated by a semi-colon. An example of this would be:

**color: blue;**

where color is the property name and blue is the property value.

Which elements these styles are applied to is specified outside of curly braces. Some common options are "**tag**" if the style is applied to every element of a particular tag, "**.class**" for elements with the class of "class", and "**#id**" for specific elements noted with the id of "id". (Notice the lack of punctuation before the tag, the "." before the class, and the # before the id.)

An example would be:

---

```
.specific-class {  
    margin: 10px,  
    border: 1px solid black,  
}
```

---

Where every element with the class "specific-class" would have a margin of 10 pixels and a border that is 1 pixel thick, solid and black.

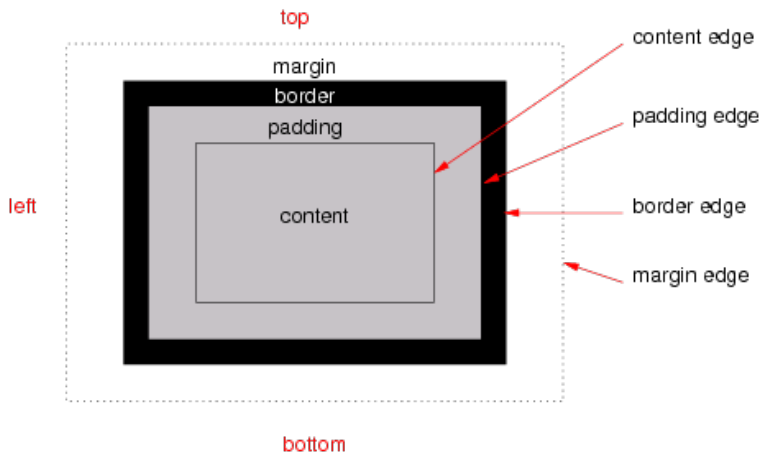
### Location of CSS

There are three options for where to place CSS code:

1. **Inline** Placed within the html code within the element. Example: `<p style="color: red;">`
2. **Internal** Placed within the style tag, usually within the head tag. Example: `<style>p { color: red;}</style>`
3. **External** Placed within an external .css file. This is then linked via the link tag as seen in 2.1.1.

This also effects how certain elements will override others. For example, any inline style will override any internal or external style, and any internal style will override any external style. The code is also read top to bottom, so styles at the bottom will override styles at the top.

## CSS Box



(W3 2020)

Every element on a webpage is contained in the box above. The outer most layer, margin, is the space outside the border. This will change the perception of how far away the box is from other boxes. After the margins and border, is the padding, this is the space between the border and the content.

### 2.1.3 JavaScript

#### Introduction to Javascript

JavaScript is a scripting language that is commonly used in web browsers (but has expanded to be able to do much more with the help of other programs such as NodeJS).

#### Declaring Variables

There are two options when declaring variables depending on if it is mutable or not. If the variable cannot be modified after initialization, use the keyword **const** followed by the variable name. For example **const number = 2**. If the variable is mutable, then use the keyword **let**. An example for this would be **let letter = 'L'**, where letter can be modified at a later time.

#### Declaring and Calling Functions

To declare a function, first use the function keyword, followed by the name, and a list of parameters within parenthesis. The internal code of the function is then placed within curly braces. To return a value to the calling function, simply use the keyword **return**. Example:

---

```
function addTwoNumbers(num1, num2) {
    return num1 + num2;
}
```

---

Then to call this function, use **addTwoNumbers(230, 384)**.

#### Window and Document Objects

Two useful objects within JavaScript is the window and document objects.



The window object references the entire browser window. This allows access to properties such as the location, history, height, width, etc. This would be equivalent to `<html>`.

The document object is a child of the window object. This contains everything that is displayed on the website. There are document methods that allow you to get an element by its id ( `getElementById()` ), as well as by its class ( `getElementsByClass()` ), and many more. This would be equivalent to `<body>`

### 2.1.4 Vue.js

#### Introduction to Vue

Vue.js is a framework that allows the creation of user interfaces. It works with HTML/CSS/JavaScript to provide re-usable components, templates, state handling, and much more. (VueJS n.d.[b])

#### Templates

Within Vue, templates can be created. These allow for a reduction in code since there is much HTML that is common between web pages. Then when these templates are rendered, different data can be inserted.

Syntactically, Vue uses double curly braces to represent areas where data will be inserted later. `{{ name }}` will be replaced with the value of the name variable. (VueJS n.d.[c])

For a simple example of where templating can reduce the amount of code written, refer back to 2.1.1. Each html document you create has to have these pieces of code. Instead if you create a template as follows:

---

```
<html>
  <head>
    <title>Title</title>
    <script src="{{_jsFile}}"></script>
    <link rel="stylesheet" type="text/css" href="cssFile">
  </head>
  <body>
    <h1>{{ title }}</h1>
    <p> {{ paragraph }}
  </body>
</html>
```

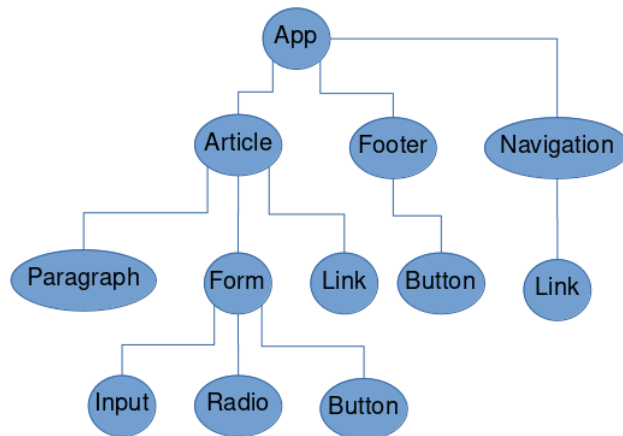
---

This could then be reused for many different websites with different information by supplying different values for the variables title, paragraph, jsFile, etc.. This of course can be expanded to much more complicated websites where there are many of the same pages. Consider Facebook, where everyone has their own profile page. These pages are standardised across the website to look the same, but the information is different on each one. A template is made once, then whenever the page is rendered it is filled with the appropriate data.

This then becomes even more powerful when responsive web apps are considered. Templates can be created for each style of device (small mobile, tablet, desktop). Then when the user visits the web app, the appropriate template can be used, all while the data remains the same.

## Components

Components are re-usable snippets of code that abstract away some code. Pre-defined options (or "props") can also be passed to the components to affect what is rendered. Many components can then be built together to form much larger apps, using minimal code compared to pure HTML/CSS/JS. For Example:



In this small example, the link component and the button component are reused within two separate parts. This is incredibly small compared to many web apps, which would reuse the components many, many times. This reduces repetition of code. (VueJS n.d.[a])

## 2.2 Practical Tasks

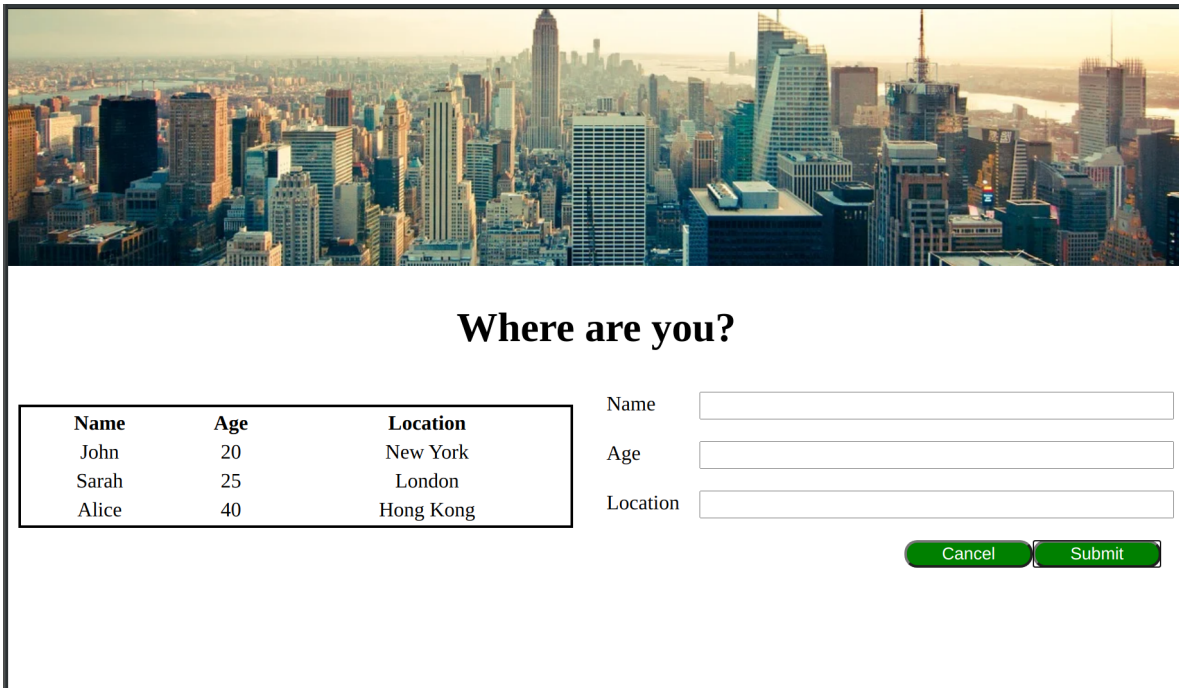
### 2.2.1 1 - HTML

This provided a very basic example to help learn the bare bones of HTML, and its commonly used tags. It also allowed practice in creating both a table and form.

#### Code

The code can be viewed on [github](#) .

## Screenshot



Name	Age	Location
John	20	New York
Sarah	25	London
Alice	40	Hong Kong

Name

Age

Location

## 2.2.2 2 - CSS

This task focussed on introducing CSS, which makes the website look attractive. It allows practice with the structure of CSS, as well as with properties that are commonly used.

Building upon the last task, I formatted the website using CSS. This allowed me to create a website that has two columns. One which contains the table of entries, the other that contains the form to input new entries. The image at the top of the page is then modified to look slimmer and stretch the width of the window, while the title is centred below.

## Code

See code on [Github](#).

## Screenshot

See 2.2.1 since these tasks are combined into one website.

## 2.2.3 3 - JavaScript

I made a small JavaScript function that takes the inputs within the form, and dynamically updates the table. I also created a separate website to calculate the inputted student scores. This helped practice modifying the DOM dynamically with JavaScript, which allows for a seamless experience as the user, since information or elements can be modified without refreshing the page.

## Code

See code for [the input form](#) and for [student grades](#).

## Screenshot

Before form submission:

## Where are you?

Name	Age	Location
John	20	New York
Sarah	25	London

Name

Age

Location

After form submission:

Name	Age	Location
John	20	New York
Sarah	25	London
Alice	40	Hong Kong

Name

Age

Location

Before score calculation:

## Score 1 Score 2 Score 3 Total Score Average Score

80	40	85		
17	63	22		
85	90	76		

After score calculation:

## Score 1 Score 2 Score 3 Total Score Average Score

80	40	85	205	68.33
17	63	22	102	34.00
85	90	76	251	83.67

## 2.2.4 4 - Vue.js

A small Todo app with three default items of a classic grocery list. New tasks can be added through the input text box. Tasks can be clicked to complete them.

This task introduced the concept of components. The component `<todo-item>` was reused multiple times, showing the utility of components. From this it shows if they are this beneficial in smaller applications, they must be exponentially so in large applications, making them incredibly important.

## Code

See the code on [Github](#).

Screenshot

## To Do:

- ☒ ~~Vegetables~~
- ☒ ~~Cheese~~
- ☐ Milk

## 2.3 Project

### 2.3.1 What I completed this week

This week I have read through the **Assignment 1: Guidelines and Rubrics**. I have understood what is required of the assignment, and started to make notes of ideas/requirements that will help me to complete this assignment.

### 2.3.2 What I will complete next week

Next week I will do research and think of an idea for my project. I expect this to take a fair bit of time to find a good idea that can be original, as well as managed within the time limitations of the course. This should take me approx. 4 hours.

## Chapter 3

# Useful Code Snippets

### 3.1 CSS

#### 3.1.1 Common Properties

color	Color of text.
background-color	Color of element background.
height	Height of element.
width	Width of element
font-size	Size of font (can be in pixels, rem (relative font size), or pre-set (small, medium, etc.))
font-family	Specify the font, or family of fonts. Multiple can be specified, with the earlier fonts taking priority.
position	How the element is positioned in the document. Some possible values: relative(Follows the standard flow), absolute(placed in a particular location in the document), fixed (placed in a particular location on the screen.)
left, right, top, bottom	How far from the left/right/top/bottom the element is.
text-align	How the text within a p or h# tag are aligned (left, center, right)
border	How the border of the element looks. Uses the value format of " <b>size style color</b> "

(Mozilla n.d.[a])

#### 3.1.2 Center an Element

To horizontally center any element:

---

```
element {  
    position: relative;  
    left: 50%;  
    transform: translateX(-50%);  
}
```

---

Any element can also be centred horizontally (by using top, and translateY) or both (using top, left and translate(X, Y)). This moves the element half way across the screen, and then moves the element back half of its width. The end effect being that it is centred on the screen. (W3 Docs n.d.)

## 3.2 JavaScript

### 3.2.1 Window.onload = function() ...

Upon fully loading the window, the provided function will then be called. This ensures the elements referenced within the function will be able to be found within the DOM. This is particularly useful with Vue, since there needs to be a rendered element for Vue to be attached. (Mozilla n.d.[b])

### 3.2.2 Round to Specific Decimal Place

To round to a specific number of decimal places, use the `toFixed(places)` method on a number variable. For example:

---

```
let number = 9.43667;  
number = number.toFixed(2);  
// number == 9.43;
```

---

(Mozilla n.d.[c])

# Bibliography

- HTML/Tabellen/Aufbau einer Tabelle* (n.d.). selfhtml. URL: [https://wiki.selfhtml.org/wiki/HTML/Tabellen/Aufbau\\_einer\\_Tabelle](https://wiki.selfhtml.org/wiki/HTML/Tabellen/Aufbau_einer_Tabelle) (visited on 07/16/2021).
- Mozilla (n.d.[a]). *CSS Tutorial*. Mozilla. URL: <https://www.w3schools.com/Css/> (visited on 07/18/2021).
- (n.d.[b]). *GlobalEventHandlers.onload*. Mozilla. URL: <https://developer.mozilla.org/en-US/docs/Web/API/GlobalEventHandlers onload> (visited on 07/18/2021).
  - (n.d.[c]). *Number.prototype.toFixed()*. Mozilla. URL: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Number/toFixed](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/toFixed) (visited on 07/18/2021).
  - (n.d.[d]). *The HTML5 input types*. Mozilla. URL: [https://developer.mozilla.org/en-US/docs/Learn/Forms/HTML5\\_input\\_types](https://developer.mozilla.org/en-US/docs/Learn/Forms/HTML5_input_types) (visited on 07/16/2021).
- VueJS (n.d.[a]). *Vue Guide - Components*. VueJS. URL: <https://v3.vuejs.org/guide/component-basics.html> (visited on 07/18/2021).
- (n.d.[b]). *Vue Guide - Introduction*. VueJS. URL: <https://v3.vuejs.org/guide/introduction.html> (visited on 07/18/2021).
  - (n.d.[c]). *Vue Guide - Template*. VueJS. URL: <https://v3.vuejs.org/guide/template-syntax.html> (visited on 07/18/2021).
- W3 (Dec. 22, 2020). *CSS Box Model Module Level 3*. W3. URL: <https://www.w3.org/TR/css-box-3/> (visited on 07/16/2021).
- W3 Docs (n.d.). *How to Horizontally Center a Div with CSS*. W3 Docs. URL: <https://www.w3docs.com/snippets/css/how-to-horizontally-center-a-div-with-css.html> (visited on 07/16/2021).
- W3 Schools (n.d.). *HTML <link> Tag*. W3 Schools. URL: [https://www.w3schools.com/Tags/tag\\_link.asp](https://www.w3schools.com/Tags/tag_link.asp) (visited on 07/16/2021).
- WHATWG (July 14, 2021). *HTML Living Standard*. WHATWG. URL: <https://html.spec.whatwg.org/multipage/syntax.html#the-doctype> (visited on 07/18/2021).