**Musicon**

In this project, you will be updating the website of an online musical instruments store, Musicon. You will incorporate your knowledge of HTML, CSS, JavaScript, and Handlebars to make a stylish multi-page interactive website!

Musicon will have three separate web pages: a home page, a store page, and a contacts page. The home and store pages already have most of the HTML and CSS set up. Your job is to make the following changes:

* Create a navigation bar using unordered lists, anchor tags, and class attributes.
* Style the website using your knowledge of display, positioning, color, and font declarations.
* Build out semantic templates using a client-side templating engine: [Handlebars](https://handlebarsjs.com/).
* Display relevant information using JavaScript objects, arrays, and control flow alongside built-in Handlebars iteration and control flow helpers.

If you get stuck during this project or would like to see an experienced developer work through it, click “**Get Help**“ to see a **project walkthrough video**.

**Tasks**

**0/42Complete**

Mark the tasks as complete by checking them off

**Creating the navbar**

**1.**

You should familiarize yourself with the structure of the project before you get started. Browse the files in the code editor: **index.html**, **store.html**, **contact.html**, **public/style.css**, and **public/main.js**.

* Examine the structure of the HTML files to see the layout of the web pages.
* Check over what content goes on each page.
* Look over how **public/style.css** is separated into sections.
* Read the properties of the context object.

Knowing the contents of the documents will help you understand the reasoning behind the tasks.

**2.**

Navigate to **index.html**. Notice that there isn’t a navigation bar.

* Locate the <nav> element.
* Add a <ul> element within the nav element.
* Assign the newly created <ul> element a class of navbar.

Hint

Make sure you include both the opening and closing tags for the <ul> element and indent it with two spaces. Place the class attribute within the opening <nav> and <ul> tag:

<nav class="name">

<ul class="anotherName"></ul>

</nav>

**3.**

Inside the <ul> element you just created, insert an <li> element. Nest an <a> element inside the <li> element.

Hint

Make sure the <a> element is nested within the <li> element, which is nested within the <ul> element.  
Your code will look like:

<ul>

<li>

<a></a>

</li>

</ul>

**4.**

Assign the <li> element a class attribute of "current".

The class of "current" has a ruleset in **public/style.css** which will highlight the <li> element. Therefore, this highlight feature will appear after you link **public/style.css** in the next assessment!

**5.**

Give the <a> element an href attribute with the value "index.html". Then include the text Home between the opening and closing <a> tags.

Hint

Place the href attribute within the <a> tag:

<a href="url">Text goes here</a>

**6.**

Below the <li> element, create another <li> element that also has a nested <a> element.

Hint

The entire <ul> element will look like:

<ul>

<li class="current">

<a href="index.html">Home</a>

</li>

<li>

<a></a>

</li>

</ul>

**7.**

In the <a> element nested in the second <li> element, and add a href attribute with the value "store.html". Insert Store into the content of the second <a> element.

Save your progress and check the browser again. You should now see two links in your navbar.

**8.**

Below the second <li> element, create another <li> element that also has a nested <a> element.

**9.**

In the <a> element you just created, add the href attribute with the value "contact.html". Also, insert the text Contact as content.

Great, you should now have a navbar in **index.html**, let’s add a navbar to **store.html**.

**10.**

Copy the entire <ul> element and navigate to **store.html**. Paste the <ul> element, with the .navbar class, inside the <nav> element.

* Delete the class attribute from the first <li> element.
* Give the second <li> element a class of "current".

Save your progress and visit the Musicon store page to see your added navbar.

Hint

The only <li> element that should have a class attribute is the second one, which has the text Store.

**11.**

Navigate to **contact.html**. Inside the <nav> element, paste in the whole <ul> element that you copied from **index.html**.

* Delete the entire class attribute from the first <li> element.
* Give the last <li> element the .current class.

Save your progress. Check the browser to see working navbars in all three pages!

Hint

The only <li> element that should have a class attribute is the last one, which has the text Contact.

**Adding CSS**

**12.**

Now, it’s time to add some style to your website. Navigate to **index.html**, add a <link> element before the closing <head> tag so that we can use a style sheet to change the look of the home page.

* Add the href attribute to the <link> element and set it to be "public/style.css".
* Add the rel attribute to the same element and set it to be "stylesheet".

Save your progress, go to the home page in the browser to see the updated style!

Hint

The <link> element is self-closing, so it is unnecessary to add a closing <link> tag. The complete <link> element will look like this:

<link rel="value" href="url">

**13.**

Navigate to **store.html**, follow the same steps as the step above to add a <link> element before the closing <head> tag to incorporate the same style sheet into the store page.

Hint

For example:

<link rel="value" href="url">

**14.**

Navigate to **contact.html**, add the same <link> element before the closing <head> tag to incorporate the same style sheet to change the look of the contact page.

Save your progress to see the updated styles on all three pages.

**15.**

Take a look at the Musicon home page. The current <section> elements that have a class attribute of container are currently positioned to the left and don’t have any margin. It would look better if you centered those <section>s and added some spacing.

Navigate to **public/style.css**. First, set a 90% width declaration for .container elements. Then add a rule to center the .container class using margin.

Hint

To select an element with a specific class, use the . character before the name of class.

.className {

/\* Rules go here \*/

}

To center an element, set the width property first, then create a declaration for margin: 0 auto; The first value, 0, assigns the top and bottom margins. The second value, auto, assigns the left and right margins. auto allows the browser to calculate how much margin to provide to the element. In this case, using auto would center the element.

selector {

width: \_\_\_;

margin: 0 auto;

}

**16.**

The .container elements are now centered but the text isn’t. Since you’re only targeting the text in the "#introduction" elements, turn your attention to the #introduction ruleset.

Add a declaration to the #introduction ruleset to make the text centered.

Hint

To make text centered use the text-align property and set it to center.

selector {

text-align: center;

}

**17.**

Now you should add some spacing to the #introduction ruleset so that elements don’t feel so squished together.

* Add a top margin of 15%
* Add a bottom margin of 50%.

Save your progress to see a home page that has proper spacing and centered elements.

Hint

Use margin-top to add a top margin. Use margin-bottom to add a bottom margin.

selector {

margin-top: \_\_\_;

margin-bottom: \_\_\_;

}

**18.**

Now, on the browser, navigate to store page. As it is now, it’s hard to differentiate the individual <article> elements with a class of instrument.

In **public/styles.css**, add a white background color and an opacity of 0.9 to the .instrument ruleset.

Hint

You can set both opacity and color using rbga(). It takes four values, the first three determine rgb and the last value determines the alpha, aka, opacity.

selector {

background-color: rgba(\_\_, \_\_, \_\_, \_\_);

}

**19.**

Save your progress. The store page now has the instruments right up against each other. You should now add some separation between the different .instrument elements.

* Add a top and bottom padding of 2% and the left and right margin to 5%.
* Set the top and bottom margin to 5% and the left and right margin to auto.

Hint

When you assign two values to the padding or margin property the first value will set the top and bottom padding/margin, the second value will set the left and right padding/margin.

selector {

padding: \_\_\_;

margin: \_\_\_;

}

**20.**

It’s more visually appealing to have rounded borders. For the .instrument ruleset, assign the border-radius property to be 5px for the same elements.

Great work styling the .instrument cards! Save your progress to see the changes you’ve made.

Hint

border-radius defines the arc of the borders of an element. The greater the value, the more arc the borders of an element are:

selector {

border-radius: 5px;

}

**Building the templates**

**21.**

After creating the navbar and adding the styling, now it’s time to build semantic templates using Handlebars.

Navigate to **index.html**, to include Handlebars, insert <script src="handlebars.min.js"></script> right after the <link> tag within the <head> element.

You’ll notice that the Handlebars file is already placed inside the project directory, but you can also use a CDN if you choose.

Hint

Copy the html below and paste the code on the line after <link rel="stylesheet" href="public/style.css">.

<script src="handlebars.min.js"></script>

By including the code above, you’re about to access the Handlebars library.

**22.**

Navigate to **store.html**, add the same <script> element, as the previous task, on the line after the <link> CSS tag.

**23.**

Navigate back to **index.html**, you can deliver a template to the browser by including it in a <script> tag.

* Add another <script> tag below <script> tag for Handlebars.
* Give the new tag an id of templateHB.
* Add the type attribute to the same tag and set it to be "text/x-handlebars-template".

Hint

The complete <script> tag will look like this:

<script id="templateHB" type="text/x-handlebars-template">

</script>

**24.**

Next, start on a simple template for the home page.

You’ll be adding three elements inside the newly created <script> tag. First add a <h1> element, followed by a <p> element, followed by an <a> element. These elements will not be nested.

Hint

For example:

<script id="template" type="text/x-handlebars-template">

<h1></h1>

<p></p>

<a></a>

</script>

**25.**

Add Handlebars expression to the <h1> and <p> tag.

* Between the opening and closing <h1> tags, add a {{title}} expression.
* Between the opening and closing <p> tags, add a {{body}} expression.

Hint

The pair of double curly braces, {{ }}, denotes a Handlebar expression. The content within the expression is going to be filled by the property of an object. For instance, the {{title}} and {{body}} content will match the title and body property of the context object:

<h1>{{title}}</h1>

<p>{{body}}</p>

**26.**

Add the href attribute and text to the <a> tag.

* Give the <a> tag the href attribute and set it to be "store.html".
* Between the opening and closing <a> tags, add the Shop Now text.

**27.**

Inside the #introduction element, add an id of information to the .container element and delete the nested tags.

Stuck? Get a hint

**28.**

Now, navigate to **public/main.js**, familiarize yourself with the provided context object.

For the home page you’ll be using the title and body properties. Later on, you’ll be using the instruments property for the store page.

**29.**

Now it’s time to write JavaScript!

* Under the context object, declare a variable named templateElement using the const keyword.
* Assign to templateElement the result of calling document.getElementById() with an argument of "templateHB".

Hint

The document method getElementById() returns an element object representing the element whose id property matches the specified string:

const newVariable = document.getElementById("idOfElement");

**30.**

The next step in creating a Handlebars template is to get the HTML markup contained within the templateElement.

* Access the .innerHTML of templateElement and assign it to a new variable named templateSource.

Hint

Access the innerHTML element property on the templateElement by using the dot notation. The element property innerHTML gets or sets the HTML markup contained within the element:

const anotherNewVar = chosenElement.innerHTML;

**31.**

Compile a template using the Handlebars.compile() method.

* Pass the templateSource into the Handlebars.compile() method as an argument.
* Assign a compiled template returned above to a new variable named template.

Hint

Handlebars.compile() is a built-in method from Handlebars that returns a compiled template:

const template = Handlebars.compile(templateSource);

**32.**

After calling Handlebars.compile() with an argument, a function is returned to the template. template will accept an object and use the properties of the object to fill in a Handlebars template.

* Pass the provided context object into the template function as an argument.
* Assign the return value of the step above to a new variable named compiledHtml.

Hint

The compiled template returned by the Handlebars.compile() method is a function that can be called with a context object as an argument - the returned value is a string containing of HTML with a filled out template.

const compiledHtml = template(context);

**33.**

Finally, render the compiled HTML in the browser.

* Use the document.getElementById() method to get an element with an id of information on the document.
* Set the innerHTML property on the element returned above to be the compiledHtml.

Hint

It will look like this:

document.getElementById('information').innerHTML = compiledHtml;

**34.**

You just created your first templated web page! Now it is time to create your next templated web page with the skills you just learned.

Navigate back to **store.html**, create a <script> element that will incorporate Handlebars expressions.

* In the <head> element, add the <script> tag on the line after the <script> tag for Handlebars.
* Give the new <script> an id of templateHB.
* Add the type attribute to the same tag and set it to be "text/x-handlebars-template".

Hint

The complete <script> tag will look like this:

<script id="templateHB" type="text/x-handlebars-template">

</script>

**35.**

Copy the entire contents of the first <article> with class instrument. Paste the contents inside the newly created <script>.

Hint

The starting template will look like this:

<script id='template' type='text/x-handlebars-template'>

<article class="instrument">

<img class="image" src="https://s3.amazonaws.com/codecademy-content/courses/learn-handlebars/musicon/electronic-keyboard.png" alt="Electronic Keyboard">

<section class="details">

<h2 class="name">Electronic Keyboard</h2>

<p class="description">A piano welcomed to the 21th century. Pianists celebrate the compact form factor and the diversity of synthesized rhythms all in one masterful musical machine.</p>

<p class="price"><del>Price: $1,999.00</del></p>

<p class="sale">On Sale! $1,699.89</p>

</section>

</article>

</script>

**36.**

Currently, you have a template for one instrument, but the Musicon store has four instruments. Conveniently, Handlebars offers the built-in {{each}} block helper to iterate through an array.

Wrap the .instrument element in the template with the {{each}} block helper. Provide the starting {{each}} expression with an argument of instruments.

Hint

The <script> will look like:

<script id="templateHB" type="text/x-handlebars-template">

{{#each instruments}}

<article class="instrument">

<img class="image" src="images/electronic-keyboard.png" alt="Electronic Keyboard">

<section class="details">

<h2 class="name">Electronic Keyboard</h2>

<p class="description">An electronic keyboard or digital keyboard is an electronic musical instrument, an electronic or digital derivative of keyboard instruments.</p>

<p class="price"><del>Price: $1,999.00</del></p>

<p class="sale">On Sale! $1,699.89</p>

</section>

</article>

{{/each}}

</script>

**37.**

Now it’s time to replace some hard coded values with Handlebars expressions. Change the value of the src and alt attribute within the <img> tag with a {{this.image}} and {{this.name}} expression respectively.

Hint

The <img> tag will look like this:

<img class='image' src="{{this.image}}" alt="{{this.name}}">

**38.**

Replace the contents inside the .name, .description, .price and .sale elements with Handlebar expressions. Use the following expressions in their respective fields: {{this.name}}, {{this.description}}, {{this.price}} and {{this.sale}}.

Hint

The elements listed above will look like:

<h2 class="name">{{this.name}}</h2>

<p class="description">{{this.description}}</p>

<p class="price"><del>Price: {{this.price}}</del></p>

<p class="deal">After Discount: {{this.deal}}</p>

**39.**

You might notice some instruments are on sale and others are not. You can account for this using a built-in Handlebars block helper, {{if}}, which acts like the if conditional in JavaScript.

Use the {{if}} block helper to display the on-sale price. If the this.sale property is truthy, you should also display the <p> elements that have the classes price, sale and deal. Add an {{else}} section, in cause this.sale is falsy, to display the this.price element (without the nested <del> tag).

Hint

It will look like this:

<script id="templateHB" type="text/x-handlebars-template">

{{#each instruments}}

<article class="instrument">

<img class="image" src="{{image}}" alt="{{name}}">

<section class="details">

<h2 class="name">{{this.name}}</h2>

<p class="description">{{this.description}}</p>

{{#if sale}}

<p class="price"><del>Price: {{this.price}}</del></p>

<p class="sale">On Sale! {{this.sale}}</p>

{{else}}

<p class="price">Price: {{this.price}}</p>

{{/if}}

</section>

</article>

{{/each}}

</script>

**40.**

Now, locate the #showcase element. In the <section> that has a class of container, add an id of information. Since you don’t need the hard coded values anymore, delete the all the elements that have a class of instrument.

The #information <section> should be empty but the web page should still be filled with instruments!

**41.**

Great, you refactored your code to use Handlebars. Take advantage of your set up template to add a new instrument to Musicon!

Add another object in the instruments array that has the following properties:

* Set the image property to be 'https://s3.amazonaws.com/codecademy-content/courses/learn-handlebars/musicon/violin.png'.
* Set the name property to be 'Violin'.
* Set the description property to be 'A versatile that is suited for any and all occasions. Those wearing tuxedos can strum together a classic. Others who prefer overalls can call it a fiddle and play some folk songs.'.
* Set the price property to be '$245.00'.

After creating the object successfully, save your progress. You will see the violin added to the store page.

Hint

Add each property on a new line and separate them with a , and indent the properties with two spaces. It will look like this:

{

image: 'https://s3.amazonaws.com/codecademy-content/courses/learn-handlebars/musicon/violin.png',

name: 'Violin',

description: 'A versatile that is suited for any and all occasions. Those wearing tuxedos can strum together a classic. Others who prefer overalls can call it a fiddle and play some folk songs.',

price: '$245.00'

}

**42.**

Great work! The home and store pages look leagues better than when you started. If you want to challenge yourself, consider:

* Add/Remove instruments to the store.
* Change the overall layout of the website.
* Create additional styling in **public/style.css**.
* Add and link to an additional page for Musicon.