**Introduction to Handlebars**

In this lesson, you will expand your ability to create dynamic web pages by learning about an external library, Handlebars.js!

A library is a collection of code that is already written that makes development easier. In the case of Handlebars, you are provided a templating engine which allows you to generate reusable HTML with JavaScript. Another benefit is that Handlebars keeps a clear separation of when you're writing HTML or JavaScript.

Invitations are a great example of a “real life” template. Invitation cards usually include the invitee's name, the venue location, the time and date, and maybe a short description. If you had to write all of that information out, then you would realize that most of the information is repeated — the only change you really need to make is the name! Creating a template with a blank space for the name would make it much easier for you to invite all your friends and family!

The idea of templating becomes even more useful for web pages that have thousands or even millions of views. Take for instance this exercise that you're working on right now — it has the same material for every visitor, the only major difference is your own personal icon on the top right corner. You wouldn't want to create a unique page for every single visitor, use a template and sub in the values you need to.

# Implementing Handlebars

Watch the video to get an in-depth overview of the code used in the previous exercise. In case you want to watch it at a later time, [here is the YouTube link.](https://youtu.be/nsU73YAj_qo)

The major steps of using Handlebars in a project:

1. Add the Handlebars library to your project. — one option is to use a Content Delivery Network (CDN): [MDN CDN documentation.](https://developer.mozilla.org/en-US/docs/Glossary/CDN)
2. Create a Handlebars script in your HTML file.
3. In your JavaScript file, grab the HTML stored in the Handlebars script.
4. Use Handlebars.compile() to return a templating function.
5. Pass in a context object to the templating function to get a compiled template.
6. Render the compiled template onto the web page.

# Using Handlebars expressions

The power of Handlebars lies in its reusability and extensibility. Handlebars expressions allow us to accomplish this.

Inside a script, Handlebar expressions are wrapped with double braces, {{ }}. The content inside the curly braces serves as a placeholder.

For instance, if we use the following script:

<script id="foo" type="text/x-handlebars-template"> <p>{{bar}}</p> </script>

And our JavaScript file looks like:

const source = document.getElementById('foo').innerHTML; const template = Handlebars.compile(source); const context = { bar: 'Text of the paragraph element' }; const compiledHtml = template(context);

After running our code, the expression, {{bar}}is replaced with the value of context.bar in our JavaScript file. In other words, compiledHtml will contain a string of '<p> Text of the paragraph element </p>'.

# Handlebars "If" block helper

So far, you've only used Handlebars expressions to insert values in your templates. If you want to check for a specific object property before you insert a value, Handlebars provides you with the {{if}} helper block. The {{if}} helper is similar to the if conditional in JavaScript but there is a difference in syntax:

{{#if argument}} // Code to include if the provided argument is truthy {{/if}}

Notice that the example above has both opening {{#if}} expression and a closing {{/if}} expression. The code block inside the block ll be added to the final HTML template if the argument provided is truthy.

# Handlebars "Else" section

In the previous exercise, you used {{if}} to determine if the compiled HTML should include a block of code. But, if that argument turns out to be falsy then you'll just have a blank section in your HTML!

Instead, you can add a default line of code by creating an else section, using Handlebar's {{else}} expression.

Take a look at the following code to see how {{else}} is implemented:

{{#if argument}} // Code to include if the provided argument is truthy {{else}} // Code to include if the provided argument is falsy {{/if}}

In this exercise, you'll incorporate an else section into your code from the previous exercise.

# Handlebars "Each" and "This" - Part I

Another helper that Handlebars offers is the {{each}} block which allows you to iterate through an array. Just like the {{if}} block, there is an opening {{#each}} expression and closing {{/each}} expression. Inside the {{each}} block, {{this}} acts as a placeholder for the element in the iteration.

Below is an example of the Handlebars {{each}} block:

{{#each someArray}} <p>{{this}} is the current element!</p> {{/each}}

This {{each}} block would be paired with an array like:

const context = { someArray: ['First', 'Second', 'Third'] }

After compiling, the HTML will look like:

<p>First is the current element!</p> <p>Second is the current element!</p> <p>Third is the current element!</p>

In this exercise, you'll be writing your own {{each}} block and using the {{this}}expression.

# Handlebars "Each" and "This" - Part II

Using {{this}} also gives you access to the properties of the element being iterated over.

For instance, if your you're using the following array inside the context object:

const context = { someArray: [ {shape: 'Triangle'}, {shape: 'Circle'}, {shape: 'Square'} ] }

And your template looks like:

{{#each anotherArray}} <p>The current shape is: {{this.shape}}!</p> {{/each}}

After going through the steps of compiling, the finished HTML will look like:

<p>The current shape is: Triangle!</p> <p>The current shape is: Circle!</p> <p>The current shape is: Square!</p>

This exercise will provide additional practice using the {{each}} block and accessing properties using {{this}}.

# Combining "If" and "Each"

In the previous exercises, you've mostly worked with individual expressions, however, you have the ability to combine expressions! In this exercise, you will combine the {{if}} block and the {{each}} block together in a single <script>!

Let's revisit the code from the previous exercise. In this exercise, you'll be adding an element to the languages array and adding a new property, modern, to all the elements. Then you will manipulate the Handlebars script to show different texts based on the modern property.

**Review**

Great work you now know the fundamentals of using Handlebars! Let's recap a few key points:

* Handlebars is an external library used to create templates which are a mix of HTML, text, and expressions.
* Handlebars uses expressions which are wrapped inside double braces like: {{someVariable}}
* A script tag with a type of "text/x-handlebars-template" is used to deliver a template to the browser.
* Handlebar.compile() returns a templating function from a template script script intended for Handlebars.
* A template created from .compile() will take an object as an argument and use it as context to generate a string containing HTML.
* Handlebars has built in block helpers which can be included in a Handlebars script.
* Block helpers have a starting expression and an ending expression. The starting expression will have a # appears before a keyword. The ending expression will have the same keyword but with a / character to denote the end.
* The {{if}} will conditionally render a block of code.
* An {{else}} expression can be inserted into an if helper block and used as part of the conditional statement.
* {{each}} is another built-in helper expression which accepts an an array to iterate through.
* In the block helper functions, the {{this}}expression gives context and serves as a placeholder for the current value.