**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.

**Instructions**

**1.**

Take a look at the text on the web page and note all the instances that ‘vanilla’ comes up.

In **main.js**, re-assign the value of flavor in the context object to 'chocolate'. Then run your code.

Notice all the changes you made with one edit!

Hint

In context, delete 'vanilla' and replace it with 'chocolate'.

let context = {

flavor: 'chocolate'

};

const source = document.getElementById('ice-cream').innerHTML;

const template = Handlebars.compile(source);

const context = {

  flavor: 'chocolate'

};

const compiledHtml = template(context);

const iceCreamText = document.getElementById('scream');

iceCreamText.innerHTML = compiledHtml;

# mplementing 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>'.

**Instructions**

**1.**

Start by creating your Handlebars template in **index.html**. Under the opening <head> tag, create a <script> element. Give the script element an attribute of id with a value of 'greet'. Then, add another attribute, type to the same script, and assign type a value of 'text/x-handlebars-template'.

The id will allow you to select the HTML inside the script and the type determines that it is going to be used by the Handlebars object.

Hint

Make sure you’re adding the <script> element inside the content of the <head> element. A <script> element has both opening and closing tags.

**2.**

In the <script> element you just made, add a Handlebars expression with greeting inside the expression.

Hint

Handlebars expressions use double curly braces {{ }}. Then inside the expression, add the content! In this case, it is greeting. Your <script> should resemble:

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

{{greeting}}

</script>

**3.**

Now navigate to your **main.js** file. You want to access the HTML stored in the <script id="greet" type="text/x-handlebars-template"> element to later make a Handlebars template:

* Access document.getElementById('greet').innerHTML and assign it to a const variable source.

Hint

Make sure you’re working inside the **main.js** file. The syntax for this step will look like:

const source = document.getElementById('greet').innerHTML;

**4.**

source is a string that contains the raw HTML, but you want to be able to replace the Handlebars expressions with actual values. Sounds like you need a function to replace the Handlebars expressions in the HTML.

Call Handlebars.compile() with source and save the returned value in a template using const. template is a function that when passed an object returns a completed/compiled template in a string.

Hint

The syntax for this step will look like:

const template = Handlebars.compile(source);

**5.**

Create a context object to hold values for filling out the Handlebars template.

Create an object context and save it as a const variable. Add a key of greeting with the value 'Hello World!'.

Hint

To create an object, the syntax would look similar to:

const someVariable = {

key: value

}

**6.**

To replace the Handlebars expressions with the values in context, call template() with an argument of context and save the value to a const variable compiledHtml.

Hint

The syntax looks like:

const compiledHtml = template(context);

**7.**

Now you have to select the element that will contain the templated HTML.

Create a variable named fill using the const keyword and assign it document.getElementById('hello')

Hint

The syntax looks like:

const fill = document.getElementById('hello');

**8.**

It’s time to add the compiled template HTML to the web page.

Assign compiledHtml to fill‘s innerHTML property.

Once you pass the checkpoint, the browser will render ‘Hello World!’.

Hint

To access the innerHTML property, use dot notation.

fill.innerHTML

Then assign compiledHtml to fill.innerHTML

fill.innerHTML = compiledHtml;

const source = document.getElementById('greet').innerHTML;

const template = Handlebars.compile(source);

const context = {

  greeting: 'Hello World!'

};

const compiledHtml = template(context);

const fill = document.getElementById('hello');

fill.innerHTML = compiledHtml;

# 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 an opening {{#if}} expression and a closing {{/if}} expression. The code block between those expressions will be added to the final HTML template if the argument provided is truthy.

**Instructions**

**1.**

In the **index.html** file, locate the <script> that has an id of 'ifHelper'. Inside the located <script> element, add an {{if}} helper block. Provide an argument of opinion to the opening {{if}} expression.

Hint

To create an {{if}} helper block use the following syntax:

{{#if argumentHere}}

// Code to include in template if argumentHere is truthy

{{/if}

**2.**

Inside the {{if}} block, add an opening and closing <p> tag. Inside the created <p> add in the text: "The correct way to say 'GIF' is GIF!".

This template will now include a <p> element that says "The correct way to say 'GIF' is GIF!" if opinion is truthy.

Hint

Your <script> element will look something like:

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

{{#if argumentHere}}

<p>Text goes here!</p>

{{/if}}

</script>

**3.**

Go to **main.js**. Some of the boilerplate code has been provided for you. However, you still have to create the object that you want to pass into template.

In a line below template, declare a variable named context using the keyword const. Then, assign to context an object that has a property of opinion and a value of true.

Hint

The syntax will look like:

const context = {

opinion: true

};

**4.**

Finally, in the last line of **main.js**, assign debateElement‘s .innerHTML property the value of compiledHtml.

Run the code and you should see some text appear on the browser!

Hint

To access the innerHTML of an element use dot notation, like so:

selectedElement.innerHTML

const source = document.getElementById('ifHelper').innerHTML;

const template = Handlebars.compile(source);

const context = {opinion: true};

const compiledHtml = template(context);

const debateElement = document.getElementById('debate');

debateElement.innerHTML = compiledHtml;

# 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.

**Instructions**

**1.**

In **index.html**, in the <script> that has an id attribute of 'ifHelper', add an {{else}} expression inside the the {{if}} helper block.

Hint

So far, your <script> element will look like:

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

{{#if argumentHere}}

<p> The correct way to say 'GIF' is GIF! </p>

{{else}}

{{/if}}

</script>

**2.**

Great, now you’ve started on your else section, but you need to include some code. Add into your else section a <p> element that contains the text: There's no right way to say GIF!

Hint

The completed <script> will look something like:

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

{{#if argumentHere}}

<p> The correct way to say 'GIF' is GIF! </p>

{{else}}

<p>New Text goes here!</p>

{{/if}}

</script>

**3.**

Go to **main.js** and assign the opinion property of context to false.

Run your code and you should see the falsy condition appear in the browser!

Hint

Make sure you’re working in **main.js**.

Locate const context and change the value of opinion to false.

# 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.

**Instructions**

**1.**

In **index.html**, in the <script> element with an id of eachHelper, create an {{each}} helper block. Use {{#each}} to start the block and {{/each}} to end the block. Add an argument of newArray to the starting expression.

Hint

The syntax for creating an {{each}} helper block looks like:

{{#each arrExample}}

{{/each}}

**2.**

Inside the {{each}} block, add a <div> element with a class attribute of "block". The content of the <div> will be {{this}}.

Before you run the code, navigate to **main.js** and find the newArray property of the context object.

Notice that the newArray property contains an array of the letters 'a', 'b', and 'c'.

Run your code. If all went well you’ll now have 3 blocks on the webpage!

Hint

The syntax will resemble:

{{#each arrExample}}

<div class="nameOfClass">Content goes here!</div>

{{/each}}

**3.**

Just to showcase how useful {{this}} and {{each}} can be, delete the elements inside the newArray. Instead of letters, filled array with elements of numbers starting with 1 and ending in 10.

Run your code. This time, if all went well, your letters will be replaced with 10 blocks that increase from 1 to 10!

Hint

Make sure you’ve deleted the letters inside the array newArray. newArray should have a value of [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].

const source = document.getElementById('eachHelper').innerHTML;

const template = Handlebars.compile(source);

const context = {

  newArray: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

};

const compiledHtml = template(context);

const displayElements = document.getElementById('display');

displayElements.innerHTML = compiledHtml;

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

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

For instance, if 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 someArray}}

<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}}.

**Instructions**

**1.**

In **index.html**, in the <script> element with an id of languagesTemp, add an {{each}} helper block. Use {{#each}} to start the block and {{/each}} to end the block. Then in the starting expression, add an argument of languages.

Hint

The syntax for creating an {{each}} helper block looks like:

{{#each arrExample}}

{{/each}}

**2.**

Inside the {{each}} block, add a <div> element with a class attribute of "card". Inside the <div> element, add a <p> element with the text: I should learn {{this.name}}..

After compiling, for each iteration of elements, {{this.name}} will be replaced with a value.

Hint

The content for your script will look like:

{{#each languages}}

<div class="card">

<p>I should learn {{this.name}}.</p>

</div>

{{/each}}

**3.**

Navigate to **main.js**. In the context object, add a languages property and set its value to an empty array.

Hint

Make sure you’re in **main.js**. Adding another property in an object means to create another key-value pair:

const context = {

newKey: newValue

};

**4.**

Populate the languages array with 3 objects. Each object will have a name property. The values for the name are: 'HTML', 'CSS', 'JavaScript'.

Hint

Adding a property to an object means to create another key-value pair:

{

newKey: newValue

}

Separate your objects with a comma.

      {{#each languages}}

        <div class="card">

          <p>I should learn {{this.name}}.</p>

        </div>

      {{/each}}

const context = {

  languages: [

    {

      name: 'HTML'

    },

    {

      name:'CSS'

    },

    {

      name: 'JavaScript'

    }

  ]

};

# 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.

**Instructions**

**1.**

In **main.js**, add a modern property with a value of true to each element of the languages array.

Hint

To add another property to an existing object, add another key-value pair. Remember to separate the properties using a comma. Take a look at the example:

const context = {

languages: [

{

name: 'html',

newKey: newValue

}

]

}

**2.**

Now, add another object to the array in the languages property of context. This object will have a name property with a value of 'COBOL' and a modern property of false.

If you take a look at the web page, you’ll see that you should learn every language. But, since ‘COBOL’ isn’t a modern language maybe it should be given less priority. Let’s change that in the next step.

Hint

You’re adding an object with two properties to the array in the languages property. It will be very similar to the other objects already in the array.

**3.**

Navigate to **index.html**, in the Handlebars <script>, add a starting {{if}} expression directly under the <div class="card">. Add an argument of this.modern to the starting {{if}} expression.

Then, directly above the closing <div> tag, add the closing {{if}} expression.

Hint

The starting expression will have a # character before the keyword. The closing expression will have a / character before the keyword. Your code will resemble:

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

{{#each languages}}

<div class="card">

{{#if this.modern}}

<p>I should learn {{this.name}}.</p>

{{/if}}

</div>

{{/each}}

</script>

**4.**

Great, now you should see that ‘COBOL’ is no longer on the web page!

But remember, you can use {{else}} to create a default behavior for when languages are not modern.

In the Handlebars <script> element, add an {{else}} section to the {{if}} block. Inside the {{else}} section, add a paragraph element that has the text When I have time, I'll learn {{this.name}}.

Now you’ve covered both cases when the language is modern and when it isn’t!

Hint

The {{else}} section goes under the <p> element of the {{if}} block. It does not need a closing expression nor a special character before the else keyword. Your code will look like:

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

{{#each languages}}

<div class="card">

{{#if this.modern}}

<p>I should learn {{this.name}}.</p>

{{else}}

// Code for else section here

{{/if}}

</div>

{{/each}}

</script>

{{#each languages}}

        <div class="card">

          {{#if this.modern}}

            <p>I should learn {{this.name}}.</p>

          {{else}}

            <p>When I have time, I'll learn {{this.name}}.</p>

          {{/if}}

**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 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.

**Instructions**

If you want to challenge yourself:

* Add more elements and properties to the array in the languages property.
* Re-write the code in the code editor from scratch.
* Wrap an {{each} block inside an {{if}} block.
* Create a separate Handlebars script and insert the finished template into a different section of the web page.