CM1040 Web Development Week 18 Lecture Note

Notebook:	Web Development		
Created:	2020-10-13 4:07 PM	**Updated:**	2020-12-18 7:15 PM
Author:	SUKHJIT MANN		

Cornell Notes	**Topic:** JavaScript template libraries	Course: BSc Computer Science Class: Web Development CM1040[Lecture] Date: December 18, 2020

Essential Question:

What are templating engines in JavaScript?

Questions/Cues:

- What are helpers?
- What block/tag helpers
- How do we register a new Helper?
- What are the hash and/or data attributes of the options system variables?
- What are the built-in helpers that handlebars.js provides for use?
- What is the options.fn property?
- How do tell handlebars.js not to encode our output as HTML?
- What are partials in handlebars.js?
- How do include comments in handlebars.js?
- What is logging in handlebars.js?

Notes

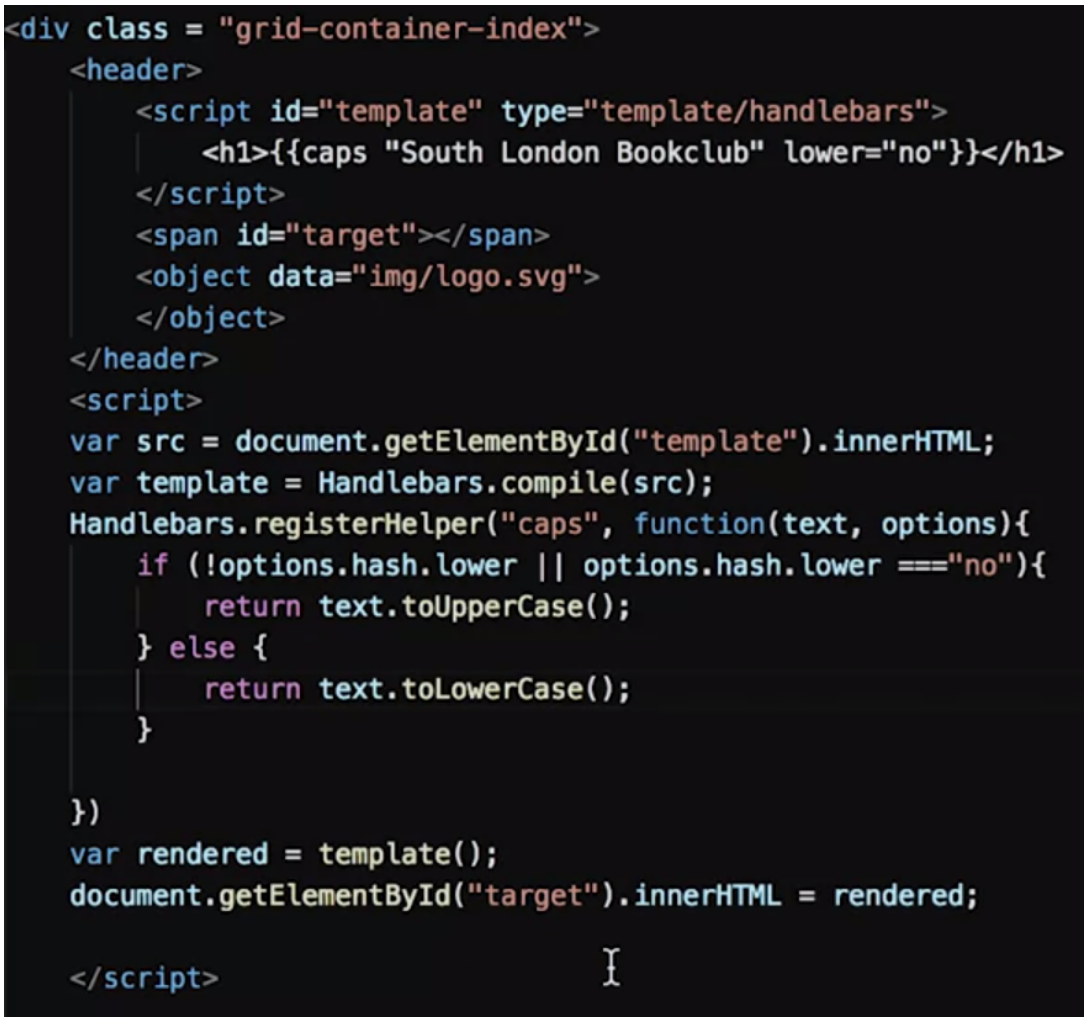
- Helpers = help to add control flow statements to our code. Helpers are functions that take data and run operation before delivering the HTML result
    - There two types of block and tag helpers for clarity
        - Block helpers are usually preceded by a hashtag
        - Tag helpers or function style helpers are the ones seen previously
        - To register a new helper, we use the registerHelper function on our handlebars object. This function accepts two variables, the name of the helper and the function that needs to be called when the tag is encountered in the template

```
<div class = "grid-container-index">
    <header>
        <script id="template" type="template/handlebars">
            <h1>{{caps "South London Bookclub"}}</h1>
        </script>
        <span id="target"></span>
        <object data="img/logo.svg">
        </object>
    </header>
    <script>
    var src = document.getElementById("template").innerHTML;
    var template = Handlebars.compile(src);
    Handlebars.registerHelper("caps", function(text){
        return text.toUpperCase();
    })
    var rendered = template();
    document.getElementById("target").innerHTML = rendered;

    </script>
```

- All the variables we place in the tags will be handed over to the function we write

```
<div id="bestSellersList">
    <h2>Best Sellers</h2>
    <script id="besttemplate" type="template/handlebars">
        {{list bestsellers}}
    </script>
    <span id="besttarget"></span>

</div>

        <script>
        var bestsrc = document.getElementById("besttemplate").innerHTML;
        var besttemplate = Handlebars.compile(bestsrc);
        Handlebars.registerHelper("list", function(list){
            var output = "<ol>";
            for (var i in list){
                output += "<li>" + Handlebars.Utils.escapeExpression(list[i]) + "</li>";
            }
            output += "</ol>";
            return new Handlebars.SafeString(output);
```

- In handlebars.js, we can also specify variable using attributes like we do in HTML tags. These attributes are called hash attributes and they one of the properties of the options system variable that gets passed into the helper function



```
<div class = "grid-container-index">
    <header>
        <script id="template" type="template/handlebars">
            <h1>{{caps "South London Bookclub" lower="no"}}</h1>
        </script>
        <span id="target"></span>
        <object data="img/logo.svg">
        </object>
    </header>
    <script>
    var src = document.getElementById("template").innerHTML;
    var template = Handlebars.compile(src);
    Handlebars.registerHelper("caps", function(text, options){
        if (!options.hash.lower || options.hash.lower ==="no"){
            return text.toUpperCase();
        } else {
            return text.toLowerCase();
        }

    })
    var rendered = template();
    document.getElementById("target").innerHTML = rendered;

    </script>
```

- a second attribute that belongs to the options variable is the data attribute, similar to the hash attribute except that we usually pass it to the template function, and not only to the helper function. This way the data is available to passed on from one helper to the next, in this case it's called a global variable

```
<div id="fpintroTitle">
    <script id="greettemplate" type="template/handlebars">
        <p>{{greetings}} South London Reading Club.</p>
    </script>
    <span id="greettarget"></span>

</div>
<script>
        var greetsrc=document.getElementById("greettemplate").innerHTML;
        var greettemplate = Handlebars.compile(greetsrc);
        Handlebars.registerHelper("greetings", function(options){
            switch(options.data.language){
                case "en":
                return "Welcome to"
                case "de":
                return "Willkommen im"
                case "fr":
                return "Bienvenue à"
                default:
                return "Welcome to"
            }

        });
        var greetrendered  = greettemplate({},{data: {language: "en"}});
        document.getElementById("greettarget").innerHTML = greetrendered;
    </script>
```

- Built-in helpers =
    - The {{#if}} helpers allows us to create if statements in our code easily. We can't conditionals to the if block, so something like {{#if x>y}} isn't allowed. In this sense, the if only check for Boolean expressions. So it can only check whether something is null, undefined, empty or zero and if it is it will return false otherwise it'll return true

```
<div id="fplinkArea">
    <script id="readtemplate" type="template/handlebars">
        {{#if beingread}}
            <div class="fplinkAreaContent">
                <h2>Books being read</h2>
                <p>There are a number of books being read online. You can join any one of the groups and read along wit
                To find out more, go to the <a href="onlineRead.html">online reads page</a>.
            </div>
        {{/if}}
    </script>
    <span id="readtarget"></span>
    <script>
            var readsrc = document.getElementById("readtemplate").innerHTML;
            var readtemplate = Handlebars.compile(readsrc);
            var readrendered = readtemplate({beingread:false});
            document.getElementById("readtarget").innerHTML = readrendered;

            </script>
```

    - {{#each}} goes through a list, or what's called an array and displays it

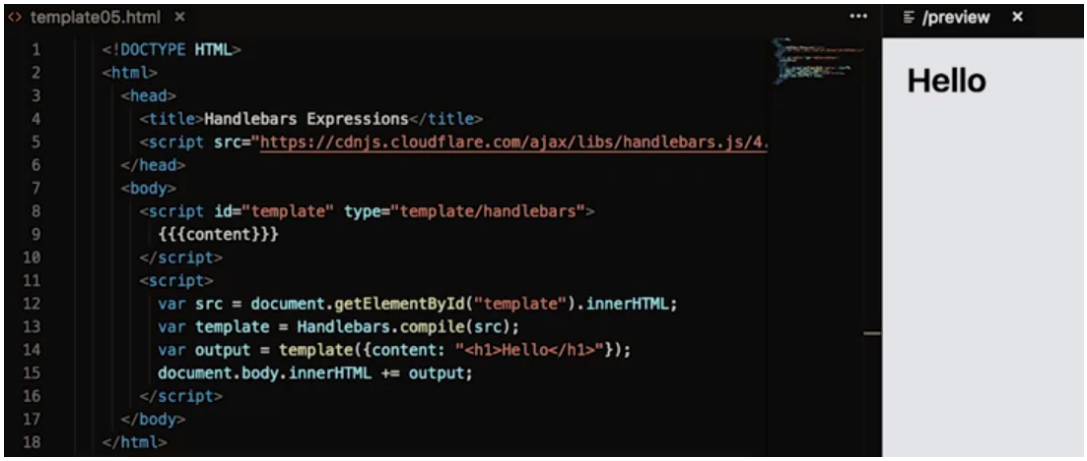
```
<div id="bestSellersList">
    <h2>Best Sellers</h2>
    <script id="besttemplate" type="template/handlebars">
        <ol>
        {{#each bestseller}}
        <li>{{this}}</li>
        {{/each}}
        </ol>
    </script>
    <span id="besttarget"></span>

</div>

        <script>
        var bestsrc = document.getElementById("besttemplate").innerHTML;
        var besttemplate = Handlebars.compile(bestsrc);
        var bestrendered = besttemplate({bestseller:["The Secret Wife by Gill Paul","The Stranger by Saskia Sarginson"
        document.getElementById("besttarget").innerHTML = bestrendered;

        </script>
```

- {{#unless}} which works in an opposite way to the if helper. It sort of the same logic as else statements
- when creating our own helpers, our helpers can use the tag syntax or we can place the name of our helper in between the curly braces to call it, or we can use block syntax like this {{#foo}} {{/foo}}
    - In the case of a block helper, anything between the block tags is considered its own function, in this sense it like any other JS function has its own variables or parameters that can be passed through
        - This is critical when we want to run our function many times with different data. A block helper can call itself with different variables by invoking another options property called fn. So when a block helpers calls something like options.fn(data(i)), it's essentially calling itself or running the part between the blocks over and over again

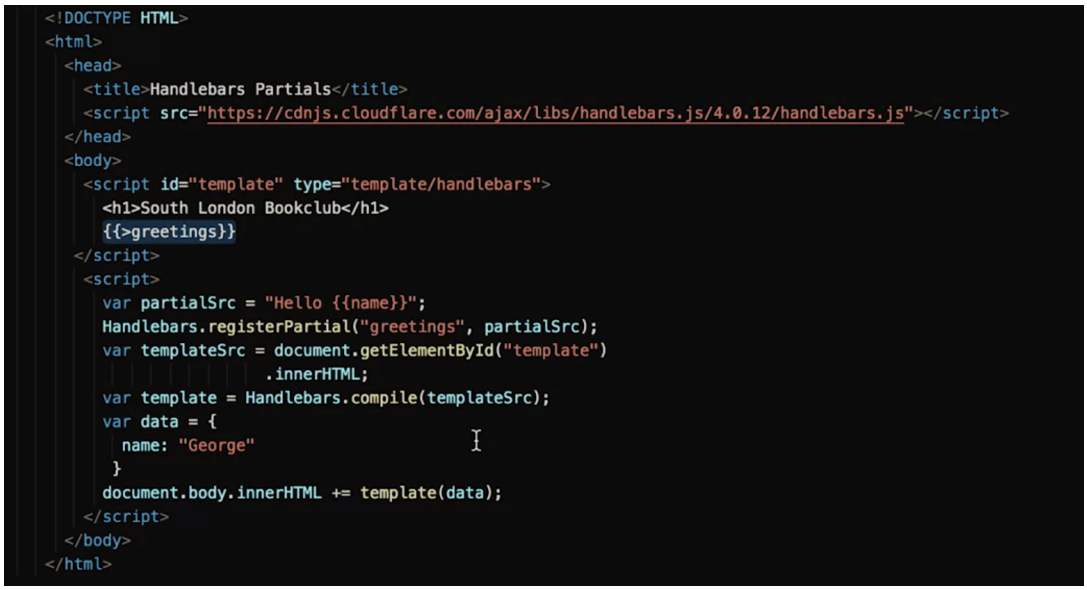
```
<body>
    <script id="template" type="template/handlebars">
    {{#bookpublished books}}
    {{name}} is {{publishedyear}}<br>
    {{/bookpublished}}
    </script>

</body>

    <script>
        var src=document.getElementById("template").innerHTML;
        var template = Handlebars.compile(src);
        Handlebars.registerHelper("bookpublished",function(data, options){
            var len = data.length;
            console.log("length: "+len);
            var returnData = "";
            for (var i=0;i<len;i++){
                data[i].publishedyear = (data[i].publishedyear < 2019) ? "published" : "not published";
                returnData = returnData + options.fn(data[i]);
                console.log("returndata: "+returnData);
            }
            return returnData
        })
```

```
        return returnData
    })
    var context = {
    "books": [{
        "name": "book1",
        "publishedyear": 2013
    }, {
        "name": "book2",
        "publishedyear": 2016
    }]
    }
    var rendered  = template(context);
    document.body.innerHTML += rendered;
</script>
```

- To have handlebars.js refrain from encoding output as HTML, we simply put another set of curly braces on the expression, so instead of two we have three curly braces, this operates similar to a safeString



- Partials = are like widgets, or like a section of the template or part of it. A partial doesn't compute data, its just a snippet of code that we can place wherever we like
    - Handlebars.js can handle partials in a few different ways depending on whether they are pre-computed, compiled or included. Regardless of this, it's key to remember that the tag we have is going to be directly replaced with a partial



```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Handlebars Partials</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/4.0.12/handlebars.js"></script>
  </head>
  <body>
    <script id="template" type="template/handlebars">
      <h1>South London Bookclub</h1>
      {{>greetings}}
    </script>
    <script>
      var partialSrc = "Hello {{name}}";
      Handlebars.registerPartial("greetings", partialSrc);
      var templateSrc = document.getElementById("template")
                        .innerHTML;
      var template = Handlebars.compile(templateSrc);
      var data = {
        name: "George"
      }
      document.body.innerHTML += template(data);
    </script>
  </body>
</html>
```

    - To include comments in handlebars.js we use this syntax : {{!-- --}}
- Logging = logging in handlebars has 4 levels referred as : debug, info, warn and error. They're ranked by urgency, with error being the most urgent, followed by warn, info and then debug.
    - When we want to log a message, we can decide its urgency on whether we think its informative or an actual error. The default level is error. If we want to change this, we have to change the Handlebars.logger.level variable and either set it to 0, 1, 2 or 3.
        - Handlebars.logger.log(urgencylevel, placeholderName) is the syntax to use the logger
        - Another way of logging is to log things inside a template with a log helper like this: {{log "Some Text or Objet"}}
            - This a good way to debug templates and discover any coding or logic errors
        - The default of the helper is 1 and that info level. But we can change this by the options data property like this:
            - var html = home{{home: Sarah"}, {data: {level: 2}}}

Summary

In this week, we learned about helpers. partials, logging, comments and so much more