JavaScript Libraries Lab-Homework (Day 16)

Objectives

- Language Libraries (lodash, Moment, Mathjs)
- Component Libraries (DataTables, Chart JS)
- Visual Libraries (React, Material, PrimeNG)
- Frameworks (Angular, Vue)

For this lab you will download several different JavaScript libraries and exercise them. Most of the operations will be simple and for demonstration purposes, but as you go through the exercises you should think about real situations where the operation could be useful. Also, when the instructions call for "outputing the result" that can be done via an alert(), a console.log(), or any other way that is comfortable. Again, this isn't a production system you just need to be able to visualize and inspect the results of the operations.

First, go to the site javascripting.com and inspect the libraries that are available. Explore each of the categories and take note of the varied capabilities that are available. Also note that some of the libraries are generic, meaning they are stand-alone and can work in any application, while others are specific to a particular framework.

Pick a couple of libraries and review their documentation. In particular, look at the following:

- Cleave
- Quill
- is
- dragula
- Sweetalert and Sweetalert2
- TinyMCE

You don't need to download these and work through the demo (unless you want to! - or have time at the end of the homework.) but it is important to browse through the documentation. And also to note if the library has any special licensing terms.

For all the exercises that follow the assumption is that the needed library will be downloaded and saved locally. If using another method (such as referring to a CDN or installing with npm) you will have to make those adjustments.
There are numerous ways to organize libraries when storing them locally. One way is to have a "js" or similar folder in each project folder and store the files that are used there. Another way is to have a central location and reference the files from there. In this lab/homework the files will be referenced from the same folder as the html file.

Also, the files will be served from the local location. That means the address in the browser will be similar to `file:///F:/DEV/web/lodash_lab.html`

Lodash

Download the lodash library. Below is the code for the template page. If you already have a standard template web page you can use that; the code I suggest to type will make use of jQuery for some references but this is not necessary - if you want to trigger the functions in some other way that is acceptable.

Template:

```
<html>
<head>
    <title>Lodash Lab</title>
    <style>

    </style>
</head>
<body>
  <p id="para">Lodash Testing</p>


  <script src="jquery/jquery-3.4.1.min.js"></script>
  <script src="lodash.js"></script>
  <script>

  $( document ).ready(function() {

    // jquery code

   });

  </script>
</body>
</html>
```

NOTE You may want to save a file like this as a template. This would allow starting with a fresh file each time you start on a new library exercise.

Next, create some values for testing. Here are the one that I have created and will base the exercises off of:

```
var arr = [1, 4, 5, 12, 55, 24, 69, 13, 5, 8, 12];
var theObj = {"id": 1, "name": "Tom Henry", "state": "KS"};
var arrObj = [
    {"id": 1, "name": "Tom Henry", "state": "TX"},
    {"id": 2, "name": "Sally Newman", "state": "FL"},
    {"id": 3, "name": "Ryan Busch", "state": "CO"},
    {"id": 4, "name": "Brenda Miller", "state": "KS"},
    {"id": 5, "name": "Ed Cage", "state": "NC"}
];
```

First exercise the lodash operations on the array. The following code is used to break the array up into smaller arrays.

```
$( document ).ready(function() {

      chunkTest();

  });

  function chunkTest()
  {
    console.log("Chunk test");
    var newArr = _.chunk(arr, 3);
    console.log(newArr);
  }
```

When the page is loaded in the browser, you will have to use the developer tools (usually F12) to see the console output. The additional exercises will encourage you to use jQuery to display the items in the browser output.

What is the output? What are the sizes of the resulting arrays? What would you use to inspect each resulting array?

Add the following to the function after the log statement:

```
    _.forEach(newArr, function(value) {
      console.log(value);
    });
```

Now what is the output?

If you haven't already, really take time take a look at what is happening there. For each element in the array, the function is being called and the element is being passed in in `value`. In this case the `value` is an array.

So now, add the code to view the individual entries of the newly formed arrays. The result should look like:

```
(3) [1, 4, 5]
 1
 4
 5
(3) [12, 55, 24]
 12
 55
 24
 ...
```

Using the `_.range()` method, create several new arrays and print out their individual elements. Then break them up with the `_.chunk()` function to create new arrays.

Create arrays with elements:

- 1-20
- 50-100
- 1-100, only even
- 1-100, only odd
- 1-500, only multiples of 20.

These can all be done in single statements.

Expand the object list to include the follow entries:

```
    {"id": 6, "name": "Will Turner", "state": "KS"},
    {"id": 7, "name": "Dawn Williams", "state": "NC"},
    {"id": 8, "name": "Sean Cunningham", "state": "NC"},
    {"id": 9, "name": "Ted Hardy", "state": "TX"},
    {"id": 10, "name": "Bruce Matthis", "state": "FL"},
    {"id": 11, "name": "Billy Smith", "state": "FL"},
    {"id": 12, "name": "Clara Jarvis", "state": "TX"},
    {"id": 13, "name": "Cindy Fisher", "state": "CO"},
    {"id": 14, "name": "Gino Donovan", "state": "CO"},
    {"id": 15, "name": "Nancy Curtis", "state": "NC"}
```

Using the documentation for the `_.filter()` function, create a statement that prints the number of resients in Colorado.

Using the documentation for the `_.countBy()` function create statements that print out the number of residents in each state.
hint - for the array returned by countby, use a foreach to print the results

Using the `_.filter()` function and any necessary string methods, find and display all the people who have a last name of 7 more letters.

Use the foreach loop and appropriate string function to print the state in all lower case. Then alter the function to print it capitalized.

Moment

Download the the moment.js file or reference it in your chosen fashion. Use the `moment()` method to create a new date; also create several other dates by parsing vaious strings as shown in the code below.

```
var mom = moment();

var bad = moment("no date");

var momISO = moment("2019-02-08 09:30");

var rfc = moment("22 Mar 2017 21:22:23 GMT");
```

Add additonal statement to print the values of the dates:

```
console.log(mom.format('YYYY MM DD'));

console.log("Bad Date: " + bad);

console.log(momISO.format());

console.log(rfc.format());
```

Read the display documentation here: https://momentjs.com/docs/#/displaying/

Using the values in the chart, print each of the above dates in the following formats:

- Long day, short month, day number, year and time
- Short month, day, year, and time
- Week of year, short year
- quarter of year, long year.

Using the difference function, find the difference between the ISO and rfc dates in days, weeks, and months. Find the difference between the current date and the rfc in weeks, months, and years.

Math

Math is a very specialized library, so here we will just enter some expressions to get a feel for how things can work. If there is a need for the advanced features of MathJS, such as statistics, probability, matrix operations, etc. the user is encouraged to evaluate those functions separately.

Using the correct math methods, calculate and display the following:

- base 10 log of 1000000
- base 2 log of 64
- base 4 log of 16384
- square root of 96
- square root of -64
- simplify the expression '3x + 4y + 9x + 2y + 8'
- the cosine of 45 degrees.

Depending on your interest, also look at the `.rationalize()` method and the other algebraic methods.

DataTable

Create a new page from the template. Add the references for the datatable. Recall that three different files are needed: jQuery, the jquery.dataTables.css file, and the jquery.dataTables.js file. Refer to the installation instructions at https://datatables.net/download/ if there are questions.

In the body of the page, create the following table:

```
<table id="person_table">
<thead>
    <tr>
      <th>ID</th><th>First Name</th><th>Last Name</th>
      <th>Cell Phone</th><th>Zip Code</th><th>State</th>
    </tr>
</thead>
<tbody>
    <tr>
      <td>111</td> <td>Brian</td><td>Smith</td>
      <td>980-996-9650</td> <td>28025</td><td>NC</td>
    </tr>
    <tr>
      <td>123</td><td>John</td><td>Doe</td>
      <td>720-859-6284</td> <td>50566</td><td>CO</td>
    </tr>
    <tr>
      <td>321</td><td>Cindy</td><td>Williamson</td>
      <td>214-933-3218</td> <td>41482</td><td>TX</td>
    </tr>
</tbody>
```

```
</table>
```

View the table to make sure the structure is correct.

Next, create the basic statement in the jQuery block that turns that into a data table. Use the column headers to sort the records and notice the reordering.

Also notice that the headings are centered while the table data is left-aligned. Using the `columns` option add a class to the cells that will center-align them.

Next, remove the above fix and run to make sure the headers are centered and the cells are left. Then, add the following style block to the top of the page:

```
table.dataTable thead th,
table.dataTable tfoot th {
  text-align: left;
}
```

This shows that table styles, defined in the jquery.datatables.css file, can be augmented manually to change the styling of the table.

Create a new page with an empty table:

```
<table id="base_table">
      <thead>

      </thead>
      <tbody>

      </tbody>
  </table>
```

Next create a JavaScript array in the script block to represent some JSON data:

```
var objs =
     [
        {
            "name":       "Jerry Nixon",
            "position":   "System Architect",
            "salary":     "$95,000",
            "start_date": "2010/03/19",
            "office":     "Scottsdale"
        },
        {
            "name":       "George Parker",
            "position":   "Director",
```

```
            "salary":     "$123,000",
            "start_date": "2002/01/19",
            "office":     "Phoenix"
        },
        {
            "name":       "Cindy Stackhouse",
            "position":   "Development Manager",
            "salary":     "$93,000",
            "start_date": "2016/04/25",
            "office":     "Dallas"
        }
    ];
```

In the `DataTable` call, use the `data` option to load the data from the variable. Note that when loading this way you'll have to specify the property name to get the data from, similar to this:

```
 "data": objs,
  columns: [
      { data: 'name' },
```

Once that is running, notice that there are no column titles. Add additional settings in each `column` to specify a title.

Finally, after the titles are in place, notice that they are centered. Instead of overridding the datatable css as we did above, create your own class:

```
<style>
.la
{
  text-align:left;
}
</style>
```

Apply that style to each column using the additional `className` setting of columns.

Chart

Much like Math, chart is a fairly specific library, but one that is very useful when needed. Create a new page from the template and reference the Chart.js file. Create a canvas area in the body, then reference that area when creating a chart:

```
var ctx = $("#chartArea"); // using jquery
var chart = new Chart(ctx,
```

```
{
    type: 'bar',
    data: {
        labels: ['XX-Small', 'X-Small', 'Small', 'Medium', 'Large', 'X-Large'],
        datasets: [{
            label: 'Planning Poker Sizes',
            backgroundColor: 'rgb(255, 55, 128)',
            borderColor: 'rgb(255, 55, 128)',
            data: [1, 2, 3, 5, 8, 13]
        }]
    },
    options: {}
});
```

Reference the properties in the ChartJS documentation and change the display to a line graph. Also, make the lines blue instead of the color above.

React

Create a new page from the template. Unlike previous examples, the files for React will be referenced from a CDN because of the "transpiler" aspect. So the references will look like this:

```
<script src="https://unpkg.com/react@16/umd/react.development.js"></script>
<script src="https://unpkg.com/react-dom@16/umd/react-dom.development.js">
</script>

<script src="https://unpkg.com/babel-standalone@6.15.0/babel.min.js"></script>
```

Those are the three scripts necessary to run/compile React on-the-fly and will suffice for the short examples done here.

For a simple React component the short example taken from the documentation is a good introduction. Inside of the `<body>` element, create a section with an id:

```
<body>
    <div id="root"></div>
```

Then at the bottom of the body, put this script:

```
<script type="text/babel">

  class HelloMessage extends React.Component {

    constructor(props) {
```

```
    super(props);
    this.state = {name: props.name};
  }
  render() {
    return (
      <div>
        Hello {this.state.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John Taylor" id="hello" />,
  document.getElementById('root')
);

</script>
```

Notice the script type parameter is *NOT* javascript - this is because the babel compiler with take this and tranform it on the fly.

Also notice that inside of the render method, the specific element is being found using the regular `document.getElementById('root')` method.

Try to replace the `getElementById('root')` with the appropriate jQuery call. What happens?

To correct the error, if you do use jQuery, you must give the render method an actual DOM element, not a jQuery object. To get the DOM element from the jQuery element, use `$("#root")[ 0 ]` and the call should work.

Now let's tie a function to the control, this will let us change the message. Make changes to the class so that it looks like this:

```
constructor(props) {
  super(props);
  this.handleClick = this.handleClick.bind(this);
  this.state = {name: props.name};
}

handleClick(e) {
  this.setState({ name: "Ted" });
}

render() {
  return (
    <div>
```

```
        <div>
          Hello {this.state.name}
        </div>
        <div>
          <button onClick={this.handleClick}>
            Change Name
          </button>
        </div>
      </div>
    );
  }
```

Although this runs, it isn't very practicle (unless your name is Ted). But in the case of the event handler, jQuery *can* be used to reference other elements on the page.
Do the following:

- Create a text input area on the page and give it an id.
- In the handleClick method, use jQuery to get the value from the text box and update the state of the React control so that it is automatically updated.

Vue

Watch the introduction to Vue video on the Vue website. Follow along with the code and in the end you should have something similar to this:

```
!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Hello Vue</title>

  </head>
  <body>

    <div id="app">
        <h1>Welcome to my first {{ frameName }} application</h1>
    </div>

    <script src="https://unpkg.com/vue"></script>
    <script>
        const app = new Vue({
            el: '#app',
            data: {
                frameName: 'Vue'
```

```
            }
        })
    </script>
</body>
</html>
```

It may be useful to come back to Vue and the Vue CLI after Angular. There will be multiple days on Angular.