

CS142 Project 3: JavaScript and the DOM

Due: Thursday, April 25, 2019 at 11:59 PM

Setup

Although this project has you run code in your browser, you need to have Node.js install on your system to run the code quality checker JSHint. If you haven't already installed Node.js and the npm package manager, follow the installation instructions ([install.html](#)) now.

Once you have Node.js installed, create a directory `project3` and extract the contents of this zip file ([downloads/project3.zip](#)) into the directory.

You can fetch the code quality tool, JSHint (<http://jshint.com/about>), by running the following command in the `project3` directory:

```
npm install
```

This will fetch JSHint into the `node_modules` subdirectory.

You will be able to run it on all the JavaScript files in `project3` directory by running the command:

```
npm run jshint
```

The code you submit should start with `"use strict";` and run JSHint without warnings.

Problem 1: JavaScript Date Picker (25 points)

For this problem, you will be implementing two interactive `DatePicker` displays (https://www.google.com/search?tbm=isch&q=datepicker&gws_rd=ssl#) using a combination of HTML, CSS, and JavaScript.

In your `project3` directory create a file `DatePicker.js` that implements a JavaScript class named `DatePicker` that can be used as in the following example:

```
var datePicker = new DatePicker("div1", function (id, fixedDate) {  
    console.log("DatePicker with id", id,  
        "selected date:", fixedDate.month + "/" + fixedDate.day + "/" + fixedDate.year);  
});  
datePicker.render(new Date("July 4, 1776"));
```

The constructor takes an argument consisting of the `id` attribute for an existing div and a date selection callback function. When a date is selected the callback function should be called with the `id` argument and an object containing the properties `month`, `day`, `year` with the number encoding of the date (e.g. `{month: 1, day: 30, year: 2016}` is the encoding of January 30, 2016).

The object should have a `render` method that takes one argument consisting of a `Date` object that selects a particular month (the object can refer to any time within the month). When `render` is invoked it replaces the contents of the date picker's div with HTML that displays a small one-month calendar such as those you might see in a travel reservation website:

- The calendar must display the days of the selected month in a grid with one line for each week and one column for each day of the week.

- Weeks run from Sunday on the left to Saturday on the right. The calendar must contain a header row displaying abbreviations for the days of the week, such as "Su", "Mo", etc.
- Each day of the month is displayed as a number.
- Some weeks may contain days not in the selected month. These days should be displayed as the number in their month, but in a dimmed fashion to indicate they are not part of the current month.
- All weeks displayed should contain at least one day belonging to the current month. Most months will display 5 weeks, but some months may display 4 or 6 depending on the days. The number of rows in your calendar should not be fixed.
- The calendar must display the name of the month and year at the top of the calendar. In addition, it must display controls such as "<" and ">" that can be clicked to change the calendar's display to the previous or next month.
- Clicking on a valid day of the current month should invoke the callback specified on the constructor with the arguments described above. Clicking on days belonging to months other than the current month should not invoke the callback.

Once you have created the JavaScript class, we have provided a file `datepicker.html` (you do not need to modify this html file) containing two empty div elements, plus a bit of JavaScript code that invokes the `DatePicker` class to display a date picker in each of the divs. One of the date pickers initially displays the current month and the other displays the month of January 2009. It should be possible to change the month of each date picker *independently* using the controls on that date picker.

The provided html file has no styling so please create a stylesheet with the filename `datepicker.css` to apply styling to the calendars and make them look nice. The corresponding link tag that requires the css file has already been added for you in the html file.

Problem 2: Simple Table Template Processing (15 points)

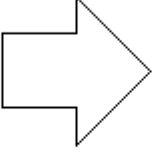
Create a file `TableTemplate.js` that implements a JavaScript class named `TableTemplate` with a **static** method `fillIn`.

Requirements for the `fillIn` method:

- This method takes three arguments consisting of the `id` attribute for a `<table>`, a dictionary object, and a string `columnName`.
- The method must first examine the header row of the table and replace any text of the form `{{property}}` with the corresponding `property` of the dictionary object. It then fills in all `{{property}}` elements in the column specified by `columnName` if there exists such a column.
- The method assumes that the first row of the table holds the column names. You are guaranteed that the column names of the input table are unique.
- If no `columnName` argument is specified, the method should process the entire table.
- If the specified `columnName` is not matched, the method should return without replacing any text in the columns. Note that you should still replace template strings in the header row regardless of whether the specified `columnName` is matched.

For example if you find a `td` element in the matching column (assuming `columnName` is specified) that has the text string "The date is the `{{day}}` day of month `{{month}}` of the year `{{year}}`" and the dictionary object argument is `{month: "January", day: "30", year: "2016"}`, you should update the text of the `td` to be "The date is the 30 day of month January of the year 2016".

Calling `TableTemplate.fillIn("table", dict, 'Part Number')`, where "table" is the table on the left and `dict` is a dictionary of strings, should generate the table on the right:

<code>{{PartNumber}}</code>	<code>{{Length}}</code>		Part Number	Length
<code>{{n14926}}</code>	<code>{{n47}}</code>		14926	<code>{{n47}}</code>
<code>{{n773}}</code>	<code>{{n3_5}}</code>		773	<code>{{n3_5}}</code>
<code>{{n9318}}</code>	<code>{{n10}}</code>		9318	<code>{{n10}}</code>
<code>{{n3045}}</code>	<code>{{n4}}</code>		3045	<code>{{n4}}</code>

original table filled-in table

If the template specifies a property that is not defined in the dictionary object the template should be replaced with an empty string. Your system need only handle properly formatted templates. Its behavior can be left undefined in the case of nested templates (e.g., `{{foo {{bar}}}}`) or unbalanced `{{`). You do not need to handle nested tables or complex table cells.

You should use your `cs142-template-processor.js` solution from project 2 to help you implement the `fillIn` method. See the script tag ordering in the html file in the `project3` directory to see how your `cs142-template-processor.js` code would be loaded.

Beware that browsers insert a `<tbody>` element around all of the `<tr>` elements in a table, if the table doesn't already contain a `<tbody>` .

Once your function has processed the entire table you should examine the `visibility` property of the table's style and if it is `hidden` update it to be `visible` .

Once you have created the JavaScript class, open the file `cs142-test-table.html` in your browser. This file represent an HTML page containing a sample template table that that will run your code and shows what your output should look like. Do not modify this file.

Additional Requirements, Hints, etc.

- You may not use jQuery or any other JavaScript library package in this project (or anywhere else in CS 142). If you were developing a real website then you should definitely take advantage of existing JavaScript libraries, but for this class we want you to learn about the low-level foundational JavaScript/DOM features.
- You may, however, use any JavaScript built-in functions. For example, you may find the following built-in functions useful in this project:
 - Methods of the `Date` class.
 - `parseInt` , `parseFloat` , `isNaN` .
- You may find some of the following DOM element properties useful in this project:
 - `cells`
 - `firstChild`
 - `nextSibling`
 - `rows`
 - `tagName`
 - `textContent`
 - `innerHTML`
- The JavaScript function `console.log` is very useful for debugging. It takes a string argument, which it prints to the JavaScript log. You can display this log with "Tools -> JavaScript console" in the Chrome control menu. If you are having trouble figuring out what is happening in your JavaScript, sprinkle `console.log` statements around your code so you can see which code is being executed.
- You may also find the `debugger` statement useful in debugging. If the text `" debugger ; "` is executed in a JavaScript program, it causes the program to drop into the JavaScript debugger. This is analogous to a breakpoint, except that your code can control when it triggers.

- If nothing seems to be happening in your JavaScript code, it's possible that your code contains an error that is causing it to be aborted. To find out if this is happening, open the JavaScript console to see if there are any errors.
- The JavaScript console and Chrome DevTools will be your best friends for the remaining projects. We encourage you to spend some time this project learning how these tools work. Feel free to stop by any of the office hours for a hands-on demonstration.

| Style Points (10 points)

Your JavaScript must be clean (appropriate use of classes, no global variables other than constructor functions, etc.), readable, and JSHint warning-free.

| Deliverables

Use the standard class submission mechanism ([submit.html](#)) to submit the entire `project3` directory.

Designed by Raymond Luong for CS142 at Stanford University

Powered by Bootstrap (<http://getbootstrap.com/>) and Jekyll (<https://jekyllrb.com>) – **learn more** ([website.html](#))