Problem 1: Meme Generator [lecture example being left to discussion section]

Implement a simple web application that allows the user to type free form into text editor and on-the-fly constructs three text memes and displays them in each of three div elements. The following screenshot should be enough to illustrate how the meme generator should work.

- The first text meme is equivalent to the original text, save that all alphabetic characters alternate between uppercase and lowercase.
- The second meme capitalizes all letters, and places a clap emoji after each word.
- The third rid of all punctuation but includes everything else in lowercase, each separated by spaces.

The details of how the meme strings are synthesized isn’t all that important, as it’s just a vehicle for the problem. What is important is your ability to construct the JavaScript controller that knows how to extract the value from the relevant textarea with each input change and update each of the three divs to present the corresponding memes.

You should implement the program to code to operate with this HTML file:

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Meme Generator</title>
  <script src="meme.js" type="text/javascript"></script>
</head>
<body>
  <textarea id="source" rows="5" cols="140"></textarea>
  <hr/>
  <u>Camel Case Meme</u>
  <div id="camel-case-meme"></div>
  <hr/>
  <u>Clap Meme</u>
  <div id="clap-meme"></div>
  <hr/>
  <u>Space Meme</u>
  <div id="space-meme"></div>
</body>
</html>
```
In the starter code, you’ll want to modify BootstrapMemeGenerator to get the DOM nodes for the textarea and meme divs, saving them in variables textarea, camelMemeDiv, clapMemeDiv, and spaceMemeDiv that you can reference later. Then, modify onTextareaInput to call constructCamelCaseMeme, constructClapMeme, and constructSpaceMeme, and call embedUpdatedMeme to display the returned strings on the page. You’ll need to implement constructClapMeme, constructCamelCaseMeme and constructSpaceMeme are already implemented for you.

Problem 2: Collapsible Lists using CSS

In this problem, you will leverage CSS in order to create an interactive collapsible list, where clicking a list item will collapse (or un-collapse) any child list items. The example we provide is a textbook table of contents. By using some simple CSS to solve this problem, the amount of Javascript we need to write will be minimized. Note that in this class, you don’t need to know how to write CSS to accomplish arbitrary goals, but you should be familiar with how it work. We will guide you through the CSS for this problem.

Start by opening collapsible-list.js and implementing ConfigureCollapsibleList. This function should loop through all li nodes on the page; for each node that contains more list items (i.e. for each node that has a child ul node), you should do two things:

- Add the closed class, so the collapsible list starts off completely collapsed
- Add a click event listener, calling toggleListItem when the li is clicked

Note that given a particular node, you can get an array of children in that node that are a particular tag type (e.g. ul) by calling node.getElementsByTagName("tagname"). (This is the same as document.getElementsByTagName, except that it only returns children of node.)

You should also implement toggleListItem(e) to toggle whether a clicked list item is collapsed. If the closed class is present on the clicked node (e.target), you should remove it and add the open class. Similarly, if the open class is present, you should remove it and add the closed class. After changing the classes, you will need to stop the event from propagating upwards in the DOM. (Do you know why?)

Recall that you can add a class by calling node.classList.add("classname"), remove a class by calling node.classList.remove("classname"), and check if a class is present by calling node.classList.contains("classname").

Once you have done this, try opening collapsible-list.html. Clicking the list items won’t do anything (although you shouldn’t see any errors in the console, either). However, if you
right click an element and click “Inspect Element,” you should see that element’s class displayed in the Chrome debugger, and you should see that class change if you click the element.

To change the presentation of the page based on what classes are present, we’ll need to add some CSS. Open collapsible-list.css. To get warmed up, let’s first apply a consistent style to all list items, so that each list item has the same kind of bullet point regardless of whether it is a top-level list item or a child:

```css
li {
  list-style-image: none;
  list-style-type: circle;
}
```

Let’s take this one step further: for closed list items, let’s replace the bullet point with a right-facing arrow indicating that the list item is collapsed, and for open list items, let’s replace the bullet point with a down-facing arrow. We can override the general li styles we just created with styles targeted at a more specific selector:

```css
.open {
  list-style-type: none;
  list-style-image: url("down-arrow.png")
}
.closed {
  list-style-type: none;
  list-style-image: url("right-arrow.png")
}
```

Finally, we need to implement the collapsing functionality, so that closed list items hide their children. To do this, we want to avoid displaying child lists (i.e. ul nodes) that are inside closed list items. We can accomplish this like so:

```css
.closed ul {
  display: none;
}
```

Note: in the styles above, we could have written .closed as li.closed and it would have had the same effect. Since it’s not necessary, convention is to keep the styles more general, so that the styles would still work if you nested a ul inside of a closed element of some other type.

Voilà! With just a few lines, we’ve managed to implement a fully-functional collapsible list. As an additional question, consider: why is the .closed ul selector so handy? How would you implement list collapsing if you could not use these “descendant selectors”?