

## Section Handout #7 Solutions

---

### Solution 1: Meme Generator

```
function BootstrapMemeGenerator() {

  /*
   * Local variables that persist, because all are referenced
   * by the inner functions, all of which are installed as
   * callbacks for mouse and input events.
   */
  let textarea = document.getElementById("source");
  let camelMemeDiv = document.getElementById("camel-case-meme");
  let clapMemeDiv = document.getElementById("clap-meme");
  let spaceMemeDiv = document.getElementById("space-meme");

  /**
   * Function: onTextareaInput
   * -----
   * Triggers every time there's any time the text within the
   * source textarea changes. This is an opportunity to build the
   * three different meme strings and embed them in the relevant <div>
   * tags. Note that this function is installed as an event handler,
   * and it references the four variables defined above.
   *
   * The benefit of this new closure-oriented approach is that we
   * only need to call document.getElementById once for each of the
   * four DOM elements ever manipulated by the code.
   */
  function onTextareaInput(e) { // e is ignored
    let text = textarea.value.trim();
    embedUpdatedMeme(camelMemeDiv, constructCamelCaseMeme(text));
    embedUpdatedMeme(clapMemeDiv, constructClapMeme(text));
    embedUpdatedMeme(spaceMemeDiv, constructSpaceMeme(text));
  }

  /**
   * Function: embedUpdatedMeme
   * -----
   * Accepts a reference to the div element which should be cleared and
   * updated to contain a new meme, the text of which is supplied via
   * content.
   */
  function embedUpdatedMeme(div, content) {
    while (div.childNodes.length > 0) div.removeChild(div.lastChild);
    let text = document.createTextNode(content);
    div.appendChild(text);
  }
};
```

```
* Function: constructClapMeme
* -----
* Returns the supplied text mostly as is, except that
* all letters have been capitalized, and a handclap emoji
* is appended to the end of each word.
*/
function constructClapMeme(text) {
  let meme = "";
  let wasInSpace = true;
  for (let i = 0; i < text.length; i++) {
    let ch = text.charAt(i).toUpperCase();
    if (ch === " " || ch === "\t") {
      if (!wasInSpace) meme += "👏";
      wasInSpace = true;
    } else {
      wasInSpace = false;
    }
    meme += ch;
  }
  return meme;
};
}
```

## Problem 2: Collapsible Lists using CSS

Javascript:

```
/**
 * Function: toggleListItem
 * -----
 * Triggers the selected li node to either expand or collapse. Accepts an
 * event "e", where "e.target" is the li node that was clicked.
 */
function toggleListItem(e) { // e.target must be a collapsible list
  if (e.target.classList.contains("open")) {
    e.target.classList.remove("open");
    e.target.classList.add("closed");
  } else {
    e.target.classList.remove("closed");
    e.target.classList.add("open");
  }
  // Stop the event from propagating upwards, causing the click handler
  // on parent list items to toggle this list item *again*. (If you
  // messed this up, top-level <li>s would work fine, second-level <li>s
  // would not be openable because two toggles cancel each other out,
  // third-level <li>s would work fine, and so on.)
  e.stopPropagation();
}
```

```
/**
 * Function: ConfigureCollapsibleList
 * -----
 * Configures all nested lists to be initially collapsed, and adds a click
 * listener to each list item that has a child list.
 */
function ConfigureCollapsibleList() {
  let nodes = document.getElementsByTagName("li");
  for (let i = 0; i < nodes.length; i++) {
    if (nodes[i].getElementsByTagName("ul").length > 0) {
      nodes[i].classList.add("closed");
      nodes[i].addEventListener("click", toggleListItem);
    }
  }
}
```

CSS:

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

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

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

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