# COMP 2406A
# Winter 2020 - Tutorial #2

---

## Objectives

- Practice solving problems with Javascript
- Practice creating DOM event handlers
- Practice manipulating the DOM
- Practice using basic HTML and CSS

## Expectations

- The minimum problems you should complete for this tutorial are problems 1-5. Problems 6 and 7 are included for an extra challenge. There are also many other ways in which you could enhance the functionality if you want more practice. Remember to use the available resources (w3schools, lecture slides, Eloquent Javascript book, etc.) for more information if you are struggling to complete the problems.

---

## Problem 1 (Creating a To-Do List Web Page)

In this tutorial, you will be building a client-side application that allows the user to create and manage a 'to-do list'. This will allow the user to add new items, remove selected items, highlight/un-highlight selected items, and sort the items.

To start, create a todo.html file. When opened, this web page should have:

1. A textbox to enter the names of new items
2. An 'Add Item' button
3. A 'Remove Items' button
4. A 'Toggle Highlight' button
5. A 'Sort Items' button

Remember to give the various components you add to the page unique IDs, which will allow you to find them using the document.getElementById method. For the time being, these buttons shouldn't do anything. We'll be adding functionality to the page throughout the rest of the tutorial.

When the page initially loads, there should be no items present on the page. However, for development and testing purposes, it may be useful to create an `onload` handler that adds items to the list automatically, so that you will not need to type new items in whenever you want to test a new version. You can remove this handler once you are finished so the list is blank when it loads.

Tips before moving forward:

1. As the complexity of your programs continues to grow, it can be important to build your solutions incrementally. Remember to break the problem down into a subset of separate steps and implement/test each step before moving on. For example, the next problem describes how clicking the Add button should be handled by your code. This complete process can be broken down into smaller parts, including creating the handling function, validating the input data, updating the state of the program (i.e., adding a new item), and updating the display. If you do a small amount of testing to ensure each step is working correctly before moving on to the next, you can simplify your debugging and make your life easier.
2. A fundamental concept throughout the course will involve organizing data in a way that facilitates its access and manipulation through code. Before diving into the programming of any of the tutorials or assignments, take some time to think about the design of your system. Identify what data you are working with and think about how it can be organized to make the implementation easier. The tutorials will provide some possible approaches but developing the ability to break down the data requirements of a problem and come up with the appropriate organization is important. It is recommended that you read through the entire tutorial/assignment and make some notes before beginning to code. Having an idea of the overall requirements of the system will help guide your design decisions. Making quality design decisions from the start will lead to cleaner code and prevent you from having to go back and modify a lot of code in order to add more functionality. Spending an hour designing the system before you start can save you 10 hours of refactoring and debugging in the future.

## Problem 2 (Creating an Add Handler)

Create a JavaScript file and include it in the HTML file you created in the previous problem. In your JavaScript file, implement a click handler for the ‘Add Item’ button.

When the user clicks the button and the item name is valid, a new item should be added to the page’s to-do list. This new item must have a checkbox so the item can be selected/deselected, as well as the text that was in the item name textbox when the

button was clicked. Whether duplicate items can be added to the list is left as a design decision for you to make. You should verify that the item name is at least 1 character long (i.e., no blank items allowed), and should reset the textbox to a blank state once the new item is created.

In order to complete this problem, as well as the rest of this tutorial, your JavaScript code will need to edit the HTML document's contents by accessing and manipulating the DOM. If you need a refresher on how to do this, review the lecture slides, lecture recording, and/or use w3schools' tutorial on the subject to help get you started:

https://www.w3schools.com/js/js_htmldom.asp

As previously mentioned, thinking of a plan to organize your data and code before starting is strongly encouraged. If you read through the entire tutorial, you will see that your client-side application will have to remember the following information about each entry in the todo list: the item name, whether the item is currently checked or unchecked, whether the item is currently highlighted or not, and the position of the item in the list (for sorting). Two ways in which you could organize the data for this tutorial are provided below. Consider these approaches and any of your own ideas, weigh the pros and cons of each, and decide on a final strategy before beginning to code.

**Approach #1: Store all the data on the page itself.**
This is a naive approach that many people will jump to immediately if they start programming after reading only the Add button description. It would be relatively simple to maintain a 'list' div that contains an entry for each item, and to create/add a new checkbox/text to represent each item when it is added.

This approach, however, has some drawbacks when you consider removing, highlighting, and sorting the items in the list. For example, how will you switch the order of the items when sorting? This is not to say that it cannot be done, but some thought will have to be put into how you will track the state of the system using only HTML elements.

One way you can do this is to maintain a pattern within the data you add to the page. For example, you may use Javascript to dynamically produce HTML like below for the list:

```
<div id="list">
    <div><input type="checkbox">Item #1</input></div>
    <div><input type="checkbox">Item #2</input></div>
    <div><input type="checkbox">Item #3</input></div>
</div>
```

You can then iterate over the child nodes of the div with id "list" and access the first child of those children to get the checkbox. Alternatively, you could get rid of the inner div elements and have the children of the "list" div alternate between checkboxes and text nodes. To sort, you could remove children and add them at the proper location within the "list" div's array of children. This approach is not very extensible though. If you add more data that you want to be associated with each item (e.g., problems 6 and 7 involve estimated time and color-coding of items), where will that data be stored?

**Approach #2 – Store all the data in Javascript objects and generate the appropriate HTML from these objects.**

Again, the data requirements for each todo list item in this problem involve storing an item name, checked status, highlighted status, and order in the list. So each todo list entry could be modelled using a simple Javascript object like { name: "Item #1", checked: false, highlighted: true, order: 0 }. Storing an array of these objects could give you a convenient way of accessing and manipulating the list data.

One issue then is: how do you display the data from an array of objects nicely on the screen? The answer is relatively simple. You can create a function that is responsible for clearing the existing list data from the page, reading the array of list entry objects, and generating a string containing the appropriate HTML to reflect those objects on the screen. Any time your handler functions change the state of your array (e.g., an item is added, removed, toggled, etc.), you can simply call this function to render the elements on the page again.

Another issue that will arise with this approach is how to handle the click events on the various checkboxes. One solution is to have an ID that is matched between the checkbox ID and the item object in your model (e.g., you could use the item name, if they are unique). The `onchange` event that is triggered when a checkbox is checked/unchecked by the user allows you to use the 'this' keyword within the handler function to refer to the element that was clicked by the user. So this.id, could be used to access the ID of the clicked checkbox, which will allow you to extract that element's ID and find the matching object in your model. In fact, any event handler can use 'this' to refer to the element on the page that triggered the event and any attribute of that element can then be easily accessed or modified.

This design separates the model (e.g., the data you are storing for the list items) and the view (e.g., what is displayed on the screen). This process of separating distinct components within a system is generally considered a good design practice. In this case, it will likely make your code cleaner and decrease the amount of DOM-based programming you must perform, while also making it easier to debug. For example, to

sort the items using this approach, you can simply sort the array and call the rendering function to update the page contents. There is no swapping of children elements through the DOM. In fact, you can even use the Javascript array sort method to do the sorting for you by specifying a simple comparison function.

When we separate different components of our system like this, it can also make debugging easier. In the first approach described above, when you sort the data and end up with incorrect results, you are unsure of whether the sorting was done incorrectly or the elements on the page were just moved around incorrectly. Using this approach, you can easily check the two components (model and view) separately and more easily narrow down the source of bugs.

## Problem 3 (Creating a Remove Handler)

Implement a click handler for the ‘Remove Items’ button. This handler should remove any checked items from the page’s to-do list. Remember, checkboxes in HTML have a property called ‘checked’ that will allow you to determine if it is checked or not. Alternatively, you can use the `onchange` event that is triggered when a checkbox is checked/unchecked by the user to update the state of your model. The better choice will largely depend on the design decisions you have made.

## Problem 4 (Adding a Highlight Handler)

Implement a click handler for the ‘Toggle Highlight’ button. This handler should:

1. Highlight any checked list items that are not currently highlighted. The highlight can be implemented in any way you want (e.g., changing the text color, the background color, etc.)
2. Remove the highlight from any checked list items that are already highlighted.
3. Not change the highlighting of any unchecked items.

## Problem 5 (Adding a Sort Handler)

Implement a click handler for the ‘Sort Items’ button. This event handler should sort the list items in alphabetical order. Whether you sort with case sensitivity or not is up to you. The handler should not change the selected or highlighted status of any of the items. It should only re-arrange all existing items into sorted order. Alternatively, you can modify your ‘add item’ handling code to automatically maintain a sorted list. If you do this, remove the ‘Sort Items’ button, as you will not need it.

## Problem 6 (Including Estimated Time)

Modify the HTML page and JavaScript code so that the user also must enter an estimated duration for the task. This time should be displayed in some way beside the item in the list. Add a selection mechanism (radio button, drop-down list, etc.) by the Sort Items button so the user can decide to sort based on the task name or the task duration. Add a selection mechanism to allow the user to sort in ascending or descending order. Alternatively, you can keep track of the sorted status of your list (unsorted, ascending, or descending) and allow the sort button to change between ascending/descending order every time it is pressed.

## Problem 7 (Allowing Color-Coding)

Instead of toggling highlighting on/off, create a way to allow the user to set the highlighting color of selected items. This could allow the user to categorize different tasks by color. Add an additional sorting selection that sorts by color so the user can easily see tasks of the same color together. Hint: HTML has a built-in color selector element.