PER SCHOLAS

ALAB 316.3.1:
DOM Manipulation (Part Two)

Version 1.0, 6/15/23
Click here to open in a separate window.

Introduction

This lab is the second of a two-part assignment in which you will manipulate the DOM using various tools and techniques. This portion of the activity focuses on making elements dynamic and interactive using DOM events and event-driven programming techniques.

Objectives

- Manipulate the DOM using JavaScript and DOM events.

Resources

This lab uses CodeSandbox. If you are unfamiliar with CodeSandbox, or need a refresher, please visit our reference guide on CodeSandbox for instructions on:

- Creating an Account.
- Making a Sandbox.
- Navigating a Sandbox.
- Submitting a Sandbox link to Canvas.

Submission

Submit your completed lab using the **Start Assignment** button on the assignment page in Canvas.

Instructions

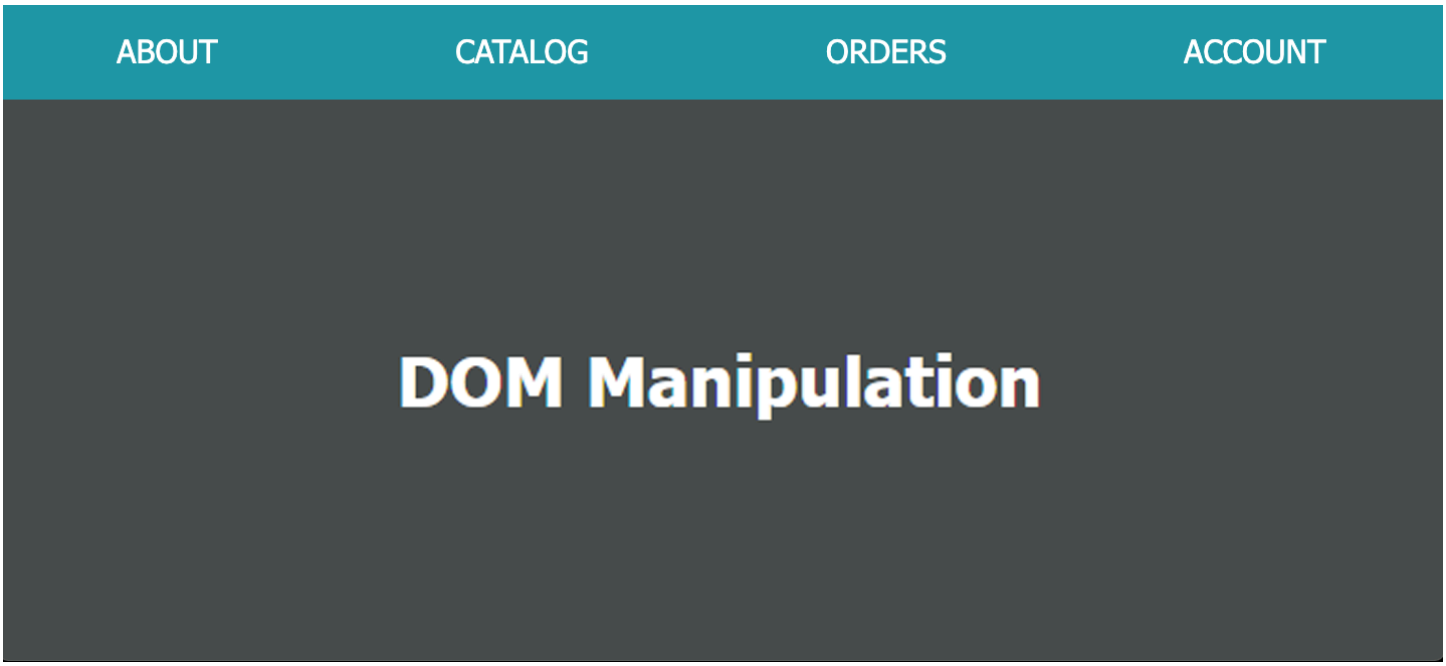
You will begin with a copy of the CodeSandbox you used for ALAB 316.1.1 - DOM Manipulation (Part One). Fork that CodeSandbox, and rename it to indicate that this is Part Two of the lab.

Do not modify any of the contents of the index.html or styles.css files. Your goal in this lab is to demonstrate DOM manipulation through JavaScript, so directly modifying the HTML or CSS files is counterproductive.

Part 1: Getting Started

Take a few moments to explore your code and refamiliarize yourself with it. Now that you have a deeper understanding of DOM manipulation concepts, if there is anything you would like to fix or change, now is the time to do so.

Progress Check - Here is what the page should look before beginning this lab:



Part 2: Adding Additional HTML and CSS

One of the most important features of an interactive user interface is feedback. The user needs to know that their actions are accomplishing something, even if it is something as simple as a button shifting color slightly or changing the cursor style when hovered, indicating that it is clickable.

In order to facilitate this, add the following additional "sub-menu" <nav> element to the index.html file within your <header> element, beneath the existing <nav> element, as follows:

```
<header>
        <nav id="top-menu"></nav>
        <!-- Add the <nav> element below -->
        <nav id="sub-menu"></nav>
</header>
```

Other than this change, **do not** modify index.html in any way.

Secondly, add the following to the styles.css file:

```
header, #top-menu {
        position: relative;
}
#top-menu {
        z-index: 20;
}
#sub-menu {
        width: 100%;
        z-index: 10;
        transition: top 0.5s ease-out;
}
#sub-menu a:hover {
        background-color: var(--top-menu-bg);
}
nav a.active {
        background-color: var(--sub-menu-bg);
        color: var(--main-bg);
}
```

Other than this change, **do not** modify styles.css in any way.

Part 3: Creating the Submenu

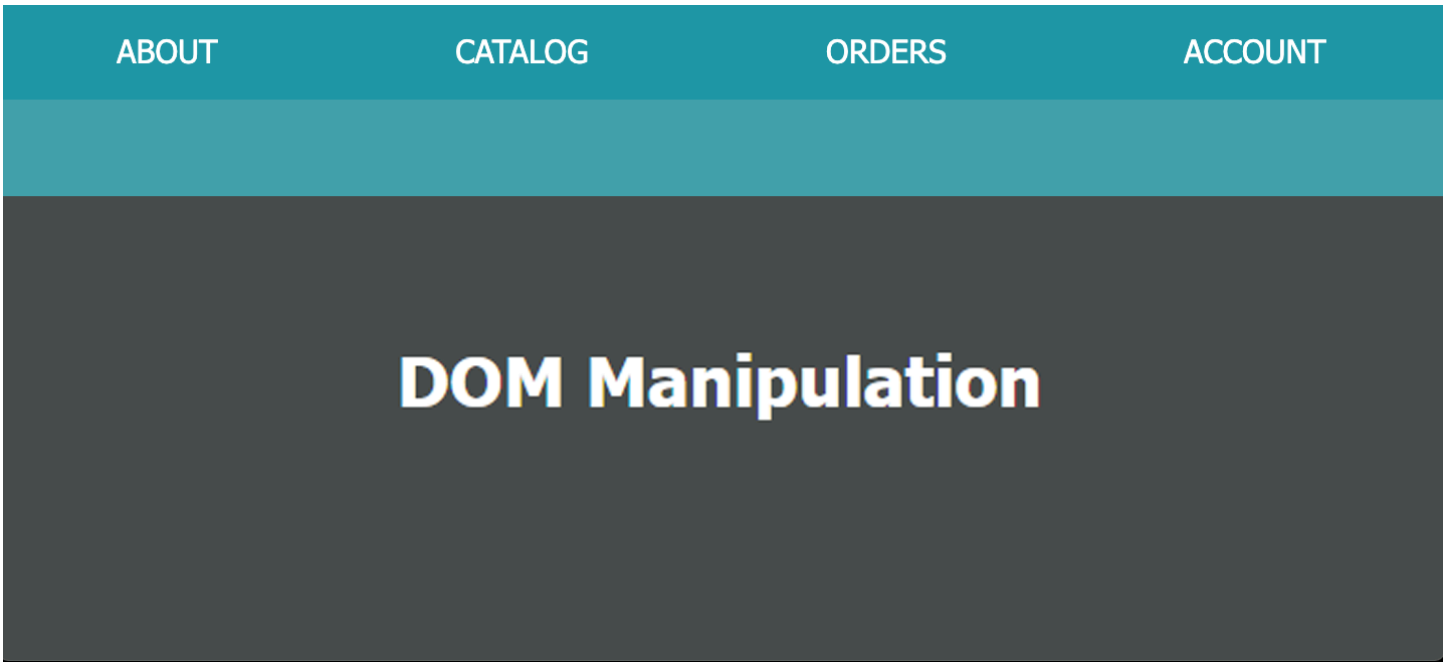
A submenu serves as an additional menu for users to select from, and offers more specific context to the top-level menu's options. We will start by using some DOM manipulation techniques to format the submenu before adding interaction to each menu component.

All future steps should be completed within the index.js file.

1. Select and cache the <nav id="sub-menu"> element in a variable named subMenuEl.
2. Set the height subMenuEl element to be "100%".
3. Set the background color of subMenuEl to the value stored in the --sub-menu-bg CSS custom property.
4. Add the class of flex-around to the subMenuEl element.

Throughout this process, note that you are also becoming accustomed to another important skill: working with another developer's code. Many of these variables, elements, CSS classes, and other features have already been developed, and you are simply working with them for your own purposes.

Progress Check - Here is what the page should look like so far:



Now, change the position of the submenu to temporarily hide it. Later, we will make the submenu appear dynamically based on user interaction:

1. Set the CSS position property of subMenuEl to the value of absolute.
2. Set the CSS top property of subMenuEl to the value of 0.

Part 4: Adding Menu Interaction

In order to add submenu links, we will need to restructure the menuLinks array within index.js. Update the menuLinks array to the following:

```
var menuLinks = [
  {text: 'about', href: '/about'},
  {text: 'catalog', href: '#', subLinks: [
    {text: 'all', href: '/catalog/all'},
    {text: 'top selling', href: '/catalog/top'},
    {text: 'search', href: '/catalog/search'},
  ]},
  {text: 'orders', href: '#' , subLinks: [
    {text: 'new', href: '/orders/new'},
    {text: 'pending', href: '/orders/pending'},
    {text: 'history', href: '/orders/history'},
  ]},
  {text: 'account', href: '#', subLinks: [
    {text: 'profile', href: '/account/profile'},
    {text: 'sign out', href: '/account/signout'},
  ]},
];
```

In order to add interaction:

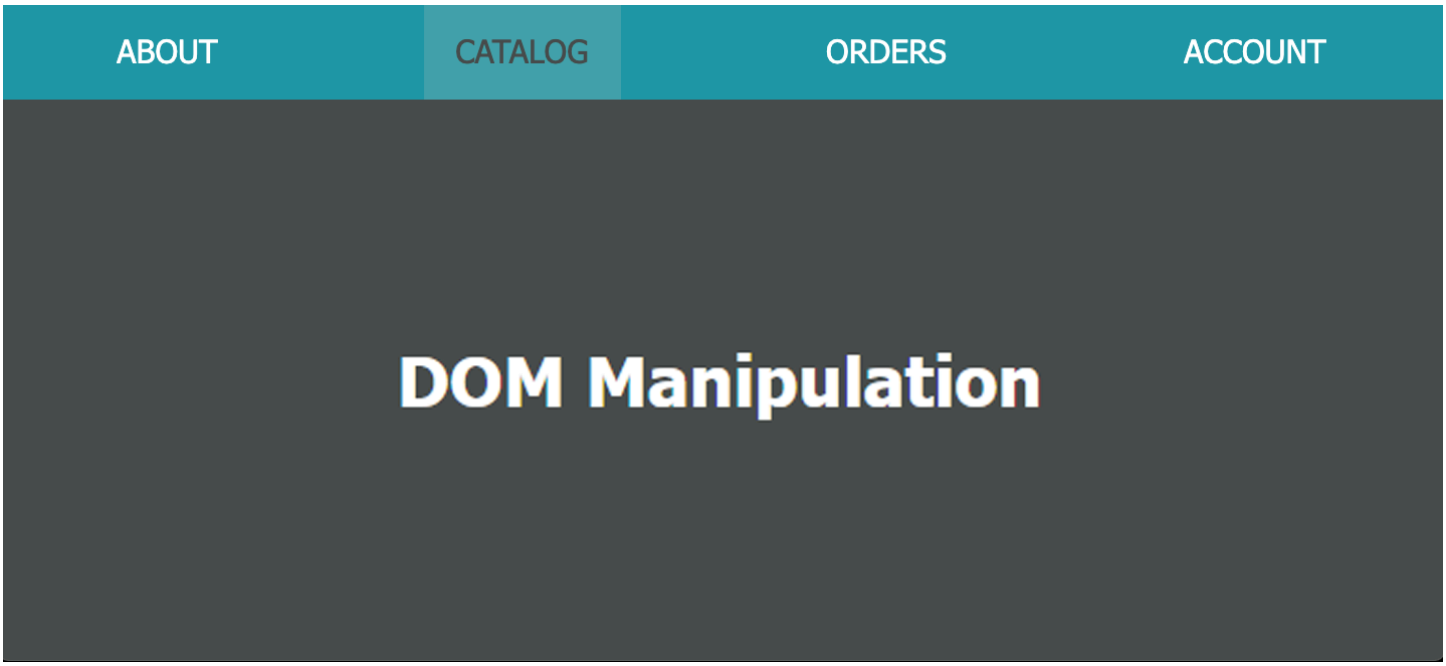
1. Select and cache the all of the <a> elements inside of topMenuEl in a variable named topMenuLinks.
2. Attach a delegated 'click' event listener to topMenuEl.
   - The first line of code of the event listener function should call the event object's preventDefault() method.
   - The second line of code of the function should immediately return if the element clicked was not an <a> element.
   - Log the content of the <a> to verify the handler is working.

Progress Check - Ensure that clicking **ABOUT**, **CATALOG**, etc. logs **about**, **catalog**, etc. when a link is clicked. Clicking anywhere other than on a link should do nothing.

Now that we have references to each of these links, and a registered event listener, we will want to add a toggled "active" state to each menu item, showing whether or not it is currently selected:

1. The event listener should **add** the active class to the <a> element that was clicked, *unless* it was already active, in which case it should remove it.
2. The event listener should **remove** the active class from each other <a> element in topMenuLinks - whether the active class exists or not.
   - **Hint:** Removing a non-existent class from an element does not cause an error!

Progress Check - Clicking any of the links should make that link active and clear the others. Clicking an active link should clear that link. Here is what it should look like so far, with "CATALOG" active:



Part 5: Adding Submenu Interaction

Within the same event listener, we want to toggle the submenu between active and non-active states. First, we will set the submenu to show or hide itself depending on the menu state:

1. Within the event listener, if the clicked <a> element does not yet have a class of "active" (it was inactive when clicked):
   a. If the clicked <a> element's "link" object within menuLinks has a subLinks property (all do, except for the "link" object for **ABOUT**), set the CSS top property of subMenuEl to 100%.
   b. Otherwise, set the CSS top property of subMenuEl to 0.
      - **Hint:** Caching the "link" object will come in handy for passing its subLinks array later.

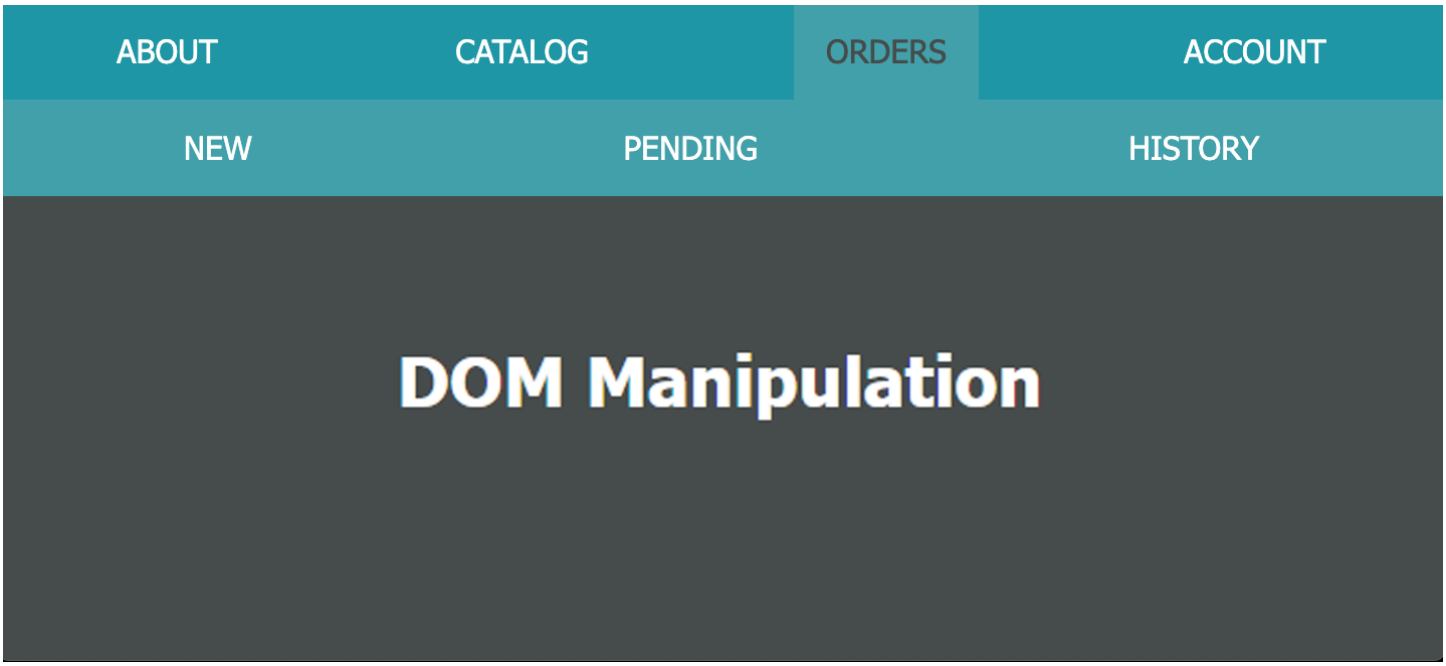
Progress Check - Ensure that clicking **CATALOG**, **ORDERS**, etc. shows the submenu bar, and that clicking them again hides it. Clicking **ABOUT** should not show the submenu bar.

The submenu needs to be dynamic based on the clicked link. To facilitate that, we will create a helper function called **buildSubmenu** that does the following:

1. Clear the current contents of subMenuEl.
2. Iterate over the subLinks array, passed as an argument, and for each "link" object:
   a. Create an <a> element.
   b. Add an href attribute to the <a>, with the value set by the href property of the "link" object.
   c. Set the element's content to the value of the text property of the "link" object.
   d. Append the new element to the subMenuEl.

Once you have created your helper function, include it in the event listener within the same logic that shows the submenu, remembering to pass the array of sub-links as an argument.

Progress Check - Here is what the page should look like so far:



The menu is almost complete! Now, we need to add interactions to the submenu items themselves:

1. Attach a delegated 'click' event listener to subMenuEl.
   - The first line of code of the event listener function should call the event object's preventDefault() method.
   - The second line of code within the function should immediately return if the element clicked was not an <a> element.
   - Log the content of the <a> to verify the handler is working.
2. Next, the event listener should set the CSS top property of subMenuEl to 0.
3. Remove the active class from each <a> element in topMenuLinks.
4. Update the contents of mainEl, within an <h1>, to the contents of the <a> element clicked within subMenuEl.
5. If the **ABOUT** link is clicked, an <h1>About</h1> should be displayed.

Part 6: Completion and Code Review

Test your menu! If it works in a way that makes sense, you have likely been successful. Your instructor has been provided with a completed version of this assignment, and time permitting, will do a brief code review so that you can make comparisons with your own approaches.

Remember, functionality is key! There are many ways to arrive at the same solution in development, and often the difference in syntax between two solutions is inconsequential. If it works, good job!

Remember to submit the link to this part of the project to Canvas using the submission instructions at the beginning of this document.