WMDD 4835 Summer 2017 Course Notes: Accessible Fly-out Menus

The accessibility techniques presented in this tutorial come from the following article from the Web Accessibility Initiative:
https://www.w3.org/WAI/tutorials/menus/flyout/

This tutorial is best done using the Chrome browser. (Since it is set up for keyboard navigation by default.) There are two HTML files included with this tutorial that demonstrate the techniques discussed herein.

Fly-out or drop-down menus are commonly used on websites to help users navigate a hierarchical site organization. The root level category links can be expanded to reveal additional links to sub-category pages.

Dropdown menus should be easily navigable using the mouse, the keyboard, and assistive technology like screen-readers:

- **Mouse navigation:** it should be easy for users with reduced dexterity to navigate the dropdown menus. Dropdown submenus should not close too quickly when the mouse leaves the submenu area.
- **Keyboard navigation:** users should be able to use the **tab** key to move through the navigation menu easily.
    - It should be visibly evident which menu item is currently in focus;
    - The user should not have to tab through every submenu item in order to access the next main menu item;
    - The user should be able to press **enter** or **space** on the keyboard to expand a submenu from the main menu item.
- **Screen readers:** It should be clear to the user that a main menu item has a submenu that can be expanded. It should be clear to the user whether the submenu has already been expanded or not.

Challenge: View and test the navigation menus on the following sites. Try using the site with both the mouse and keyboard:

- **http://bc.ctvnews.ca/**
- **http://vancouversun.com/**
- **http://www.usask.ca/**

Answer the following questions for each site:

- **How easy is it to use this site with a mouse?**
- **How easy is it to use this site with a keyboard? Is it even possible to use the keyboard?**
- **How could this site be improved to make it easier to navigate with the mouse and keyboard?**

Task 1: Write the HTML

As usual, we'll use **ul** lists within a **nav** element:

- Each **ul** list will have a **class** to label whether it is a main menu or submenu.
- Each **ul** menu will have button to expand it, marked up with an **a (anchor)** tag.
    - Each button will have a class labeling it as a button.

```
<header>
    <nav>

        <a href="" class="main-menu-button">Menu</a>

        <ul class="main-menu">
            <li><a href="#">Chickens</a></li>
            <li><a href="#">Squids</a></li>
            <li>

                <a href="#" class="sub-menu-button">Giraffes</a>

                <ul class="sub-menu">
                    <li><a href="#">Blue Giraffes</a></li>
                    <li><a href="#">Purple Giraffes</a></li>
                    <li><a href="#">Orange Giraffes</a></li>
                </ul>

            </li>
            <li><a href="#">Bees</a></li>
        </ul>

    </nav>
</header>
```

*Note: we use **anchor** tags for the menu items and buttons so that they can be focused by using the keyboard. There are other ways to make other HTML elements keyboard focusable, but lets stick with this simple method for now.

Task 2: Add ARIA attributes

It's important to make sure our menu is easy to understand and navigate while using assistive technology like screen-readers (for people who are not able to see the screen.) We can us **ARIA attributes** and **ARIA roles** to provide semantic information and context within the HTML about what each component of the site is meant for and how it can be used. For instance, it may not be clear that an anchor tag in a menu can be used to expand a submenu, because normally anchor tags are

used to navigate to a different page entirely. ARIA can help us let the user know that this anchor tag has a slightly different function.

- A description of the ARIA system can be found here: https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA
- A description of ARIA roles and attributes can be found here: https://www.w3.org/TR/wai-aria/roles

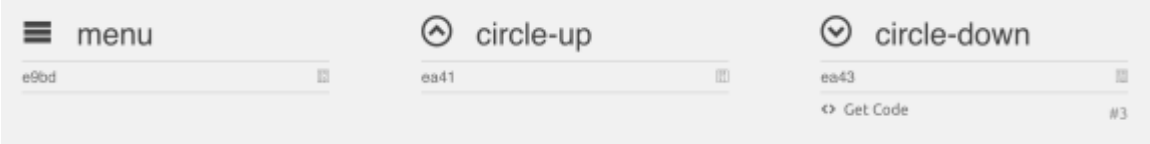
In this case, to indicate that a button is meant to expand a pop-up menu, we will add the attribute **aria-haspopup** and set the value to **true**. To indicate whether a submenu has been expanded or not, we use the attribute **aria-expanded**. Soon we will use **jQuery** to update the value of this menu as we expand or close the menu.

```
<header>
    <nav>

        <a href="" class="main-menu-button" aria-haspopup="true" aria-expanded="false">Menu</a>

        <ul class="main-menu">
            <li><a href="#">Chickens</a></li>
            <li><a href="#">Squids</a></li>
            <li>

                <a href="#" class="sub-menu-button" aria-haspopup="true" aria-expanded="false">Giraffes</a>

                <ul class="sub-menu">
                    <li><a href="#">Blue Giraffes</a></li>
                    <li><a href="#">Purple Giraffes</a></li>
                    <li><a href="#">Orange Giraffes</a></li>
                </ul>

            </li>
            <li><a href="#">Bees</a></li>
        </ul>

    </nav>
</header>
```

Task 3: Add Button Icons

We've added information to the HTML about what each component is meant for. Now we need to add some visual clues to indicate the function of the buttons. Let's use **Icomoon** for that.

Download an icon package from **Icomoon.io** with the following icons: **menu, circle-up,** and **circle-down**.



Let's link up these fonts to our page. Since we've added code to the HTML to indicate the function of our buttons, we should be able to easily link up these icons

by editing the **icomoon/style.css** style sheet, without adding any additional mark-up. (Remember to link the icomoon stylesheet in the head of your document!)

We can apply the correct icon to the correct element by selecting the buttons by their class. We can differentiate between the up and down arrow by using the CSS attribute selector to select either the **true** or **false** value for the **aria-expanded** attribute.

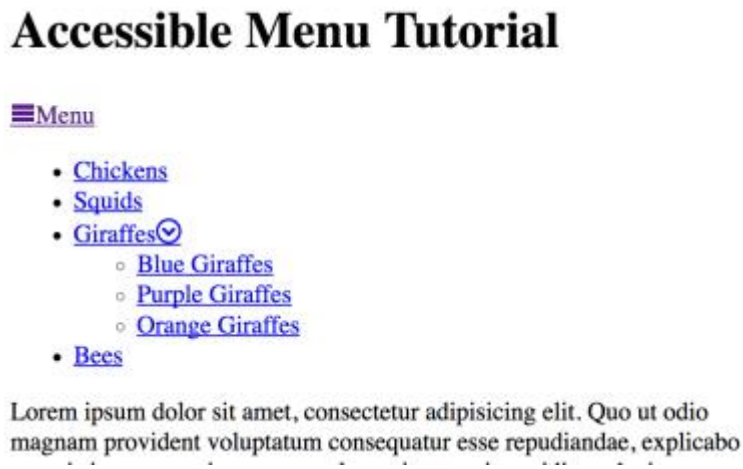
*Tip: notice that we can choose whether the icon is placed before or after the button text by using either the **before** or **after** pseudo-element.

```
/*Hamburger menu*/
.main-menu-button:before {
  content: "\e9bd";
}
/*Arrow up*/
.sub-menu-button[aria-expanded="true"]:after {
  content: "\ea41";
}
/*Arrow down*/
.sub-menu-button[aria-expanded="false"]:after {
  content: "\ea43";
}
```

We also need to apply the **icomoon font family** to the appropriate element in order to render the icons properly. To prevent the icomoon font from being applied to both the icon *and* the button text, we're selecting the **before** and **after** pseudo-elements specifically. Note also that we're selecting any **anchor** element that has the word **"button"** somewhere in the class name.

```
a[class*="button"]:before, a[class*="button"]:after {
  /* use !important to prevent issues with browser extensions that change fonts */
  font-family: 'icomoon' !important;
  speak: none;
  font-style: normal;
  font-weight: normal;
```

Our page should now look something like this:



Task 4: Adding jQuery Functionality

Dropdown menus with hover functionality are easy to use and efficient for mouse-users, but we should also add functionality to allow the menus to be expanded by clicking on the menu buttons or pressing **enter** or **space** on the keyboard when the button is focused. We can do this using **jQuery.**

To start, link the most current version of the jQuery library; from the **Google Hosted Libraries** page, copy the **jQuery 3.x snippet** and paste it into the head of your document: https://developers.google.com/speed/libraries/#jquery

Then, create a **script** element, and within that element, ensure that the page has been loaded and is ready to be processed: **$(document).ready(function(){});** . The following code should be in the head of your document:

```
<!-- jQuery Library -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<script>

    // JavaScript code goes here

    $(document).ready(function(){

        // jQuery code goes here

    });

</script>
```

Let's start by adding functionality to the *main menu button*, to display or hide the *main menu*.

First, we should create a **jQuery object** for the **main-menu-button** element, and attach an **event handler** to that object to detect the **click event**:

```
// Handle mouse clicks on the main menu button
$(".main-menu-button").on("click", function(){

    // This code gets executed when the user clicks the button

});
```

Notice that we are using the **CSS selector** for the main-menu-button class; that is, we're using a **dot** before the class name. The **.on()** method is used to indicate that we would like something to happen when the user interacts with the main menu button. In this case, we're using the "**click**" event.

- More information about **.on()** and event handling: https://www.w3schools.com/jquery/jquery_events.asp

Then, we should **toggle** a class on or off of the main-menu element. With this class (we'll use the class "**expanded**"), we can use CSS to control whether the menu is hidden or visible. We can toggle the class like this:

```
// Handle mouse clicks on the main menu button
$(".main-menu-button").on("click", function(){

    // Toggle "expanded" class on and off of the main-menu element
    $(".main-menu").toggleClass("expanded");

});
```

Here we are using the **.toggleClass("nameOfClass")** method to add or remove the class **"expanded"** from any element with the class **"main-menu**". (The class will be added if it's not already there, or removed if it *is* already there.)

*Tip: when providing the name of the class to be toggled, we do *not* use a dot!

Challenge: we could add or remove the expanded class by using the methods .addClass() and .removeClass(). Why would we choose to use .toggleClass() instead?

- For more information on **.toggleClass():** w3Schools Toggle Class

Something else we need to do is toggle the value of the **aria-expanded** attribute. When the menu is expanded, the value of this attribute should be **true**. Otherwise, it should be **false**. Unfortunately, it's not as straight-forward to toggle an attribute value as it is to toggle a class value, but we can still do it using an **if/else** statement and the jQuery **.attr()** method:

```
// Toggle the value of aria-expanded
if ( $(".main-menu-button").attr("aria-expanded") == "true")

    {
        // Change the value of the aria-exanded attribute
        $(".main-menu-button").attr("aria-expanded", "false");
    }

    else

    {
        // Change the value of the aria-exanded attribute
        $(".main-menu-button").attr("aria-expanded", "true");
    }
```

As you can see, we're using the **.attr()** method in two different ways: first we're using it to *get* the value of the attribute (to see what the value is), and then we're also using it to *set* the value of the attribute (to the opposite of what it was before).

- More information about this method can be found here: http://api.jquery.com/attr/

Task 5: Write Some CSS to Turn the Menu On or Off

We've written the jQuery code to add the expanded class to the main menu; now we just need to use CSS to actually hide or show that menu! Let's do it like this:

```
.main-menu

{
    display: none;
}

.main-menu.expanded
{
    display:block;
}
```

As you can see, we're just using the expanded class to change the **display** property from **none** to **block**.

> *Note: we should use either **display:none** or **visibility:hidden** to hide menus when they have not been expanded, rather than a visual hiding method such as **opacity:0** or **height:0**. This is because we do not want the menu to be keyboard-navigable before it has been expanded. The user should only be able to navigate to menu items that are visible on the screen.
>
> **Challenge: Extend the code we've written so far to also handle the submenu. It should just be a matter of duplicating the code we've written (both CSS and jQuery) and changing some of the class names. You should notice that when the submenu is expanded, the arrow icon automatically flips direction! That's because of the way we've set up the Icomoon stylesheet.**

Task 6: Making our jQuery Code More Robust

The jQuery code we've written so far works pretty well, but it's easy to see its limitations by creating another submenu on the page. Try copying the HTML for the current submenu and pasting it into another main menu item, like this (remember to retain the main menu item link text):

```
<nav>

    <a href="#" class="main-menu-button" aria-haspopup="true" aria-expanded="false">Menu</a>

    <ul class="main-menu">
        <li>
            <a href="#" class="sub-menu-button" aria-haspopup="true" aria-expanded="false">Chickens</a>

            <ul class="sub-menu">
                <li><a href="#">Blue Giraffes</a></li>
                <li><a href="#">Purple Giraffes</a></li>
                <li><a href="#">Orange Giraffes</a></li>
            </ul>

        </li>
        <li><a href="#">Squids</a></li>
        <li>

            <a href="#" class="sub-menu-button" aria-haspopup="true" aria-expanded="false">Giraffes</a>

            <ul class="sub-menu">
                <li><a href="#">Blue Giraffes</a></li>
                <li><a href="#">Purple Giraffes</a></li>
                <li><a href="#">Orange Giraffes</a></li>
            </ul>

        </li>
        <li><a href="#">Bees</a></li>
    </ul>

</nav>
```

When you click on one of the submenu buttons, you can see that both submenus are expanded. This is not what we want!

This is happening because currently we're creating a jQuery object that contains every element with **class="sub-menu-button"** and subsequently expanding or hiding every element with the **class="sub-menu"**. What we would like to do is only expand or hide the menu that corresponds to the button we *actually* clicked on.

We can fix the part of the code for toggling the "expanded" class like this:

```
// Select any element with a class that contains the word 'button'
$("[class*='button']").on("click", function(){

    // Toggle the class for whichever element comes immediately after
    the element that the user click on.  In this case, the ul.
    $(this).next().toggleClass("expanded");

});
```

There's a lot going on here, so let's analyze the parts.

Selecting the button: First of all, rather than selecting a specific button (like the *main-menu-button* or *sub-menu-button*), we're selecting *any* button, by using the **CSS attribute selector** with the ***=** operator within the **$()** function.

> * Tip: notice that when we use quotation marks inside of other quotation marks we need to use single quotes instead of double quotes.

Targeting the specific button that was clicked on: We're using the **this** keyword to refer to the button element that was clicked on (that is, the button that triggered this event). We use it to compose a jQuery object by using **$(this).**

- More information on **$(this)**: https://remysharp.com/2007/04/12/jquerys-this-demystified

Targeting the **ul** element that this button controls: We're using the **.next()** method in jQuery to target the element that comes immediately *after* the button that was clicked on in the HTML. This is kind of like using the + operator in a CSS selector.

- More information on **.next():** https://www.w3schools.com/jquery/traversing_next.asp

Notice that this works because the menu **ul** element comes directly *after* the button in the HTML.

Challenge: re-implement the toggling for the *aria-expanded* attribute. This should just be a matter of copying the code that you wrote before and replacing the class selector for the menu with *this*.

Challenge: You may have noticed that if you expand a submenu, hide the main menu, and then expand the main menu again, the submenu that was expanded earlier remains expanded. Maybe this is okay, but maybe we would actually like the submenu to be hidden when the main menu is hidden. Find a way to implement this. You may need to use the *.removeClass()* method and the *.find()* method (to look in an element for specific sub-elements).

- More information on **.find():** https://api.jquery.com/find/

Challenge: so far, every time we've wanted to select and modify an element in our document, we've been creating a new jQuery object by using *$()*. However, if we want to use the same jQuery object over and over, we don't need to create it every time; we can create the object once and use a variable to refer to it. For example: *var button = $(this);*. Try updating the code to use variables for the jQuery objects.

Task 7: Positioning and Animating the Menu

In a future tutorial, this is what we'll do:

- Position the menu in an attractive way
- Animate the expansion and hiding of the menus
- Implement hover functionality
- Make sure the hiding of menus is delayed, so that mouse navigation does not require precise movements.