CPSC1520 Client Side Scripting With JavaScript

In Class Lab 1

Due January: 28th, 2024 @11:59pm
Weight: 5%

Introduction

In this lab you are going to create a modal window using the dialog element. Once you complete the lab you have to upload your finished lab to Netlify. You also have to create a GitHub repository for the lab and submit the Netlify URL and the GitHub repository URL to Moodle.

Task: Call To Action Button

Take a look at the markup below this markup contains the call to action button “run speed test”. Give the button element a unique id attribute. You will use this id to create a reference variable later in your JavaScript code.

```
<header class="mb-12">
    <svg viewBox="0 0 500 500" class="w-16 h-16 mx-auto fill-indigo-600">...</svg>
    <p class="text-xs text-center font-semibold text-indigo-700 mb-3">...</p>
    <h1 class="text-center font-semibold text-indigo-700 text-3xl">...</h1>
    <p class="text-xs text-center">...</p>
    <button
        type="button"
        class="mt-3 block mx-auto rounded-lg bg-indigo-500 px-5
                py-2 font-medium text-white">
        run speed test
    </button>
</header>
```

Task: Dialog Element

Below you will find the markup for the dialog element. Give the dialog a unique id attribute. You will use the id to create a reference variable later in the JavaScript code. Not the ***open*** attribute of the dialog element . When the value appears in the dialog element the dialog box is active and you will see it.

Remove the open attribute and see the effect in the browser. By default we do not want the dialog open attribute.

```
 <main>
    <!-- email dialog (modal window) -->
    <dialog open class="backdrop:bg-blue-50/50"></dialog>
</main>
```

Task: Close Dialog Button

Below you will find the markup for the close button inside the dialog element. Give the close button a unique id attribute. You will use the id to create a reference variable later in the JavaScript code.

```
    <nav class="flex flex-end mb-8">
        <button class="ml-auto focus:outline-indigo-300 focus:bg-indigo-100
                        m-3 border-b-2 border-indigo-400">
         <svg class="w-4 h-4 fill-indigo-600 stroke-none"viewBox="0 0 24 24">...</svg>
       </button>
     </nav>
```

Task: Add JavaScript To The Web Document

Create a JavaScript file and attach it to the web document. Make sure to add the defer attribute to the script tag to help the performance of the site load. Place your script tag after the tailwindcss loading script.

```
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="icon" href="assets/icons/favicon.svg" />
    <link rel="stylesheet" href="css/tailwind.css" />
    <title>Document Title</title>
    <link rel="stylesheet" href="css/global.css" />
    <script src="https://cdn.tailwindcss.com"></script>
</head>
```

Task: Reference Variables

Create reference variables to the call to action button, the dialog element and the close button inside the dialog element. The code shown below is just an example. You need to replace the ***refVar*** with a semantic variable name that represents the functionality of each element from the DOM.

```
const refVar = document.querySelector("id-attribute");
const refVar = document.querySelector("id-attribute");
const refVar = document.querySelector("id-attribute");
```

Task: Add Event Listeners

The code shown below is just an example of adding an event listener to the call to action button and the close dialog element button.. You need to replace the ***openButtonRefVar, eventType and eventHandlerFunction*** for both the open and close buttons. Make the eventHandlerFunction names semantic reflecting the functionality or action each will perform.

```
 // call-to-action button
openButtonRefVar.addEventListener("eventType", eventHandlerFunction);

// close dialog button
closeButtonRefVar.addEventListener("eventType", eventHandlerFunction);
```

Task: Event Handler Functions

The code shown below again is provided as an example of the two event handler functions you will have to write. The first example shown below is for the call to action button. This will open the dialog element and make it active.

The second function shown is for the dialog element close button. This action will close the dialog removing it from the browser viewport.

Replace the text ***eventHandlerFuction*** with your own eventHandlerFunction name. Note that they have to match with the last task eventHandlerFunction names you provided. For example the call to action button should match with the call to action eventHandlerFunction name used in the last task.

```
// call-to-action button
function eventHandlerFunction(e) {
 dialogRefVar.showModal();
}

// close dialog button
function eventHandlerFunction(e) {
 dialogRefVar.close();
}
```

Task: Testing

Preview your work in the browser, test both buttons to make sure that the dialog element . Works as intended. If you're happy with the functionality then move to the next task.

Optional Task: Clicking Outside The Modal

By default the dialog element responds to the escape key to close the element. However a standard for closing modal windows is also to click outside the modal content on the dialog backdrop. This should also close the dialog.

Replace the text ***dialogRefVar*** with your reference variable that points to the dialog element. The event handler function in this case is provided for you. This code checks to see the coordinates of the mouse click.If the click originates outside the dialog content then the dialog element close() method is called.

```
dialogRefVar.addEventListener("click", onClickOutsideDialog);

    function onClickOutsideDialog(e) {
            const dialogDimensions = dialog.getBoundingClientRect();
       if (
         e.clientX < dialogDimensions.left ||
         e.clientX > dialogDimensions.right ||
         e.clientY < dialogDimensions.top ||
          e.clientY > dialogDimensions.bottom) {

         e.currentTarget.close();

      }
 }
```

Submission Task: Create A GitHub Repository

Create a github repository for the lab and add your files and folders. Copy the URL for the repository and submit the link to Moodle.

Submission Task: Netlify Deploy

Deploy your finished lab to Netlify. You can either drag and drop the folder or you can set up a deployment using your GitHub repository. Copy the URL that Netlify provides you with. Make sure that it is the production URL. Copy this link and submit it to Moodle.

Submission Checklist

Netlify Deployment URL as a link in Moodle
GitHub Repository URL as a link in Moodle

IMPORT PLEASE READ

If the Netlify URL that you provide generates a **404 Not Found** error or any other http error you will not receive a grade for the lab. Likewise if the URL to the GitHub repository returns a **404 error** or any other http error you will receive a grade of zero.

Both of the URLs that you submit must be links and not just a copy of the **URL** as plain text. Use the create link option provided in Moodle wysiwyg editor to create the links.