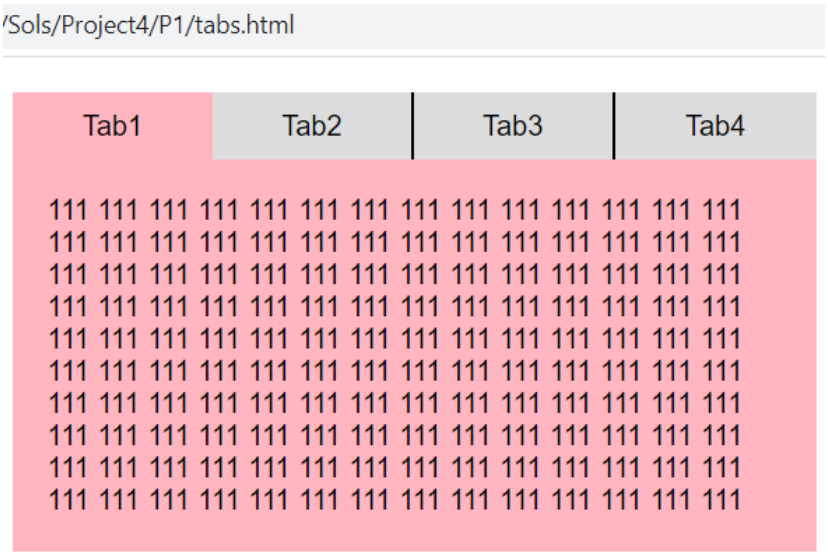
Due on November 23, 2022

Project 4

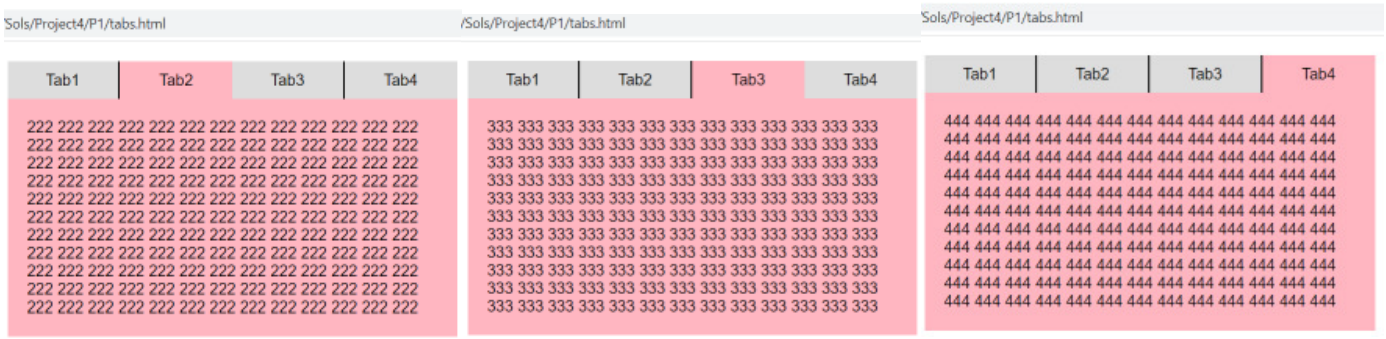
In this assignment you are asked to implement two pages using HTML, CSS and JS.

Problem 1: Implement a Tab Changer

In this problem, you are asked to implement a Component that has 4 Tabs similar to the one shown below:



The tabs must sit inside a container of width 500px centered on the window with Tab1 being active and all other tabs being in-active. The user should be able to switch to any tab by pressing the appropriate Tab area. For example, here is how the screen should look like when Tab2, Tab3 or Tab4 is active respectively:

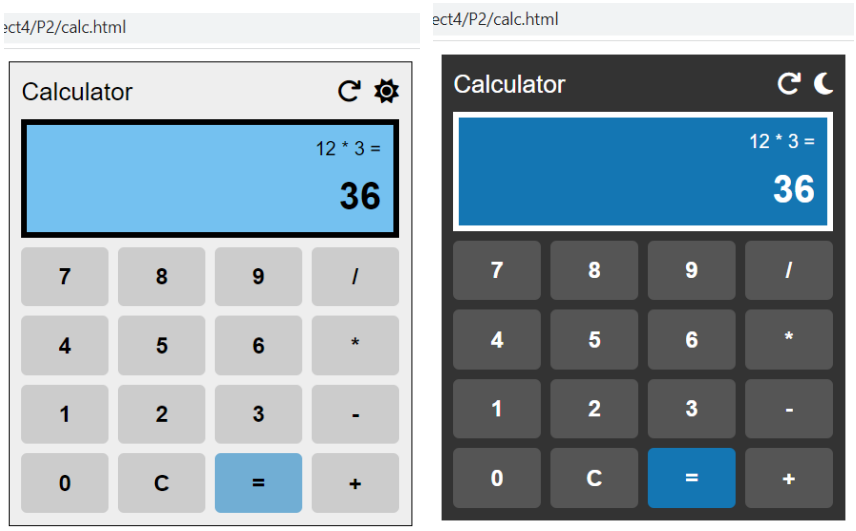


Notice how the active tab is always highlighted. Also notice the border between different Tab areas (the vertical black line that separates different tab areas). The way to implement this Tab Changer component is to have all 4 tabs inside a container. At any time, only the user-chosen Tab will be visible (the active tab) and all other tabs will be invisible. You can make any HTML element invisible by setting its display property to none, i.e., “display: none”. To make an HTML element visible, simply set its display property to a value other than “none”, e.g., block. You should keep track of the active tab and change the visibility of the appropriate tab content visible/invisible in JS.

Put all your CSS styles and JS code inside a single file named “tabs.html”.

Problem 2: Implement a Calculator

In this problem, you are asked to implement a calculator that performs basic arithmetic operations such as addition, subtraction, multiplication and division, and will have two modes of operation: A light mode and a dark mode as shown below:



The calculator must sit inside a container with a width of 360px centered on the screen. The calculator itself is a grid container with 6 rows and 4 columns (that's how I implemented it).

The first row of the grid has a height of 30px and contains the title "Calculator" on the left and two icons on the right: The first icon (arrow-rotate-right) is used to change the current operational mode and the second icon shows the current mode. The "light" mode is indicated by the "sun" icon and the "dark" mode is indicated by the "moon" icon. I used the following icons from fontawesome:

```
<i class="fa-sharp fa-solid fa-arrow-rotate-right"></i>
<i class="fa-sharp fa-solid fa-moon"></i>
<i class="fa-sharp fa-solid fa-sun"></i>
```

To use these icons, remember to include fontawesome CSS library at the head section of your HTML as follows:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.2.0/css/all.min.css" integrity="sha512-xh6O/CkQoPOWDdYTDqeRdPCVd1SpvCA9XXcUnZS2FmJNp1coAFzvtCN9BmamE+4aHK8yyUHUSCcJHgXloTyT2A==" crossorigin="anonymous" referrerpolicy="no-referrer" />
```

Notice that only one of the sun/moon icons is visible at any time depending on the operation mode. Specifically, in the light mode, we should see the "sun" icon, and in the dark mode, we should see the "moon" icon.

The second row of the grid has a height of 100px and shows the current operation begin performed and its result if it is evaluated. In the figures, the user has entered in 12 * 3 =, which evaluates to 36. The result section consists of two rows: At the top we see the current operation (12 * 3 =), and at the bottom we see the result of the current operation (36). Notice that I used different font sizes for the first and second row in the result section. Also notice that the result section has a thick border and a bluish background, which changes depending on the operation mode.

Below the result section we have a 4x4 grid with each grid cell containing one button. I set the height of each button to 50px. Also, each grid column has the same width. The buttons must be placed in the same order as shown above. The buttons must also have a hover effect. Simply set the background color of the buttons to "#999" during hover. Also notice that "=" button has a different background and hover color.

How to implement light/dark mode

The simplest way to implement the light/dark mode is the following: Create three .css files: common.css, light.css, and dark.css. common.css should contain styles that are common to both modes such as the layout, width, height etc. As you can see, the difference between the light/dark modes is in the background color of the container, the buttons and the text color: In light.css, only write the styles specific to the light mode. In dark.css, only write the styles specific to the dark mode. Then, at the head section of your html file, include common.css and light.css by default as follows:

```
    <link rel="stylesheet" href="common.css"/>
    <link rel="stylesheet" href="light.css" id="theme-css"/>
```

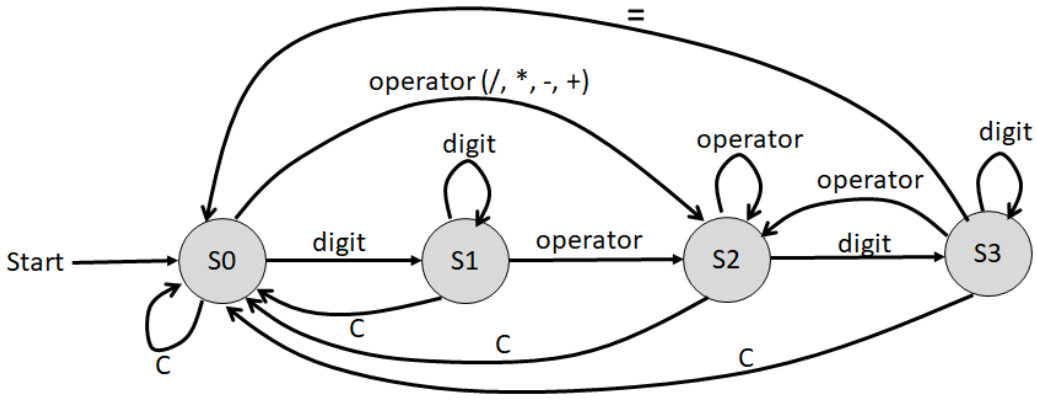
Notice that I gave an id to the HTML link element that includes light.css. When the user presses the “change mode” icon in the calculator header, you should catch this button click event in JS, then grab this link element with id=”theme-css”, and then change its href attribute. In the following example, we change the mode to “dark” mode.

```
let themeCssEl = document.querySelector('#theme-css')
themeCssEl.setAttribute('href', 'dark.css')
```

When you change the href attribute of the link element, the browser will automatically fetch dark.css file from the Web server and then apply the styles in that css file when re-rendering the page.

How to implement expression evaluation

In order to use this calculator, the user is essentially entering a math expression one char at time. You should be displaying the current expression at the first line of the result section, and the result of whatever the expression evaluates to at the bottom as shown in the figures above. In order to implement this logic correctly, I designed and implemented the following Finite State Machine (FSM) with 4 states:



When the page loads, we are at state S0 and the current value shown in the result section is 0. Regardless of which state we are in, when the user presses Clear (C) button, we clear everything and go back to S0 as seen in the FSM. In S0, if the user enters an operator, i.e., one of /, *, -, +, we set operand1 to the current value of the calculator and go to S2. If, however, the user enters a digit, we go to S1 and read in operand1 for as long as the user continues entering a digit. In S1, the user stops entering operand1 by entering an operator, which takes us to S2. In S2, the user may change the current operator for as long as s/he wants. To enter the second operand, the must enter a digit, which takes us to S3. Here, we take operand2 for as long as the user continues to enter digits. In S3, the user may stop entering operand2 either by entering a new operator or by entering ‘=’, both of which evaluate the current expression and go to their respective states. This is how I implemented the expression evaluation in my calculator. I recommend that you do the same for correctness.

Name your HTML file “calc.html”. Create 3 CSS files: common.css, light.css and dark.css as mentioned above. Put all your JS code in a file named “calc.js” and include all CSS and JS files into your HTML file.

Submission

Zip tabs.html, calc.html, common.css, light.css, dark.css and calc.js, and submit the zip file through UZEM. Do not forget to include the group member names.