**Interactive Web App Design**

Design and accessibility:

Since the web app is quite limited, only simple accessibility principles were used. There is a very simple document structure that has been adopted- just a header, and the main body of content (being the calculator) which utilises a list (ul) element for the row of buttons. It is centered in the middle of the page for minimal mouse movement. It has also incorporated some form functions in order to include an option for the user to input their own values (as opposed to something more difficult or time consuming, like clicking separate buttons to represent individual numbers or arrow directions to scale through numerical options). If there has been a problem with the input values, an error message will appear and say exactly what is causing the error so that users can have an easier time of correcting their mistakes and being able to continue using the app. Repetition has been avoided by including a ‘clear’ button, that will quickly reset the input values for the user so that they do not need to keep doing it themselves, making the process easier to use and repeat.

Visual elements to aid accessibility have also been implemented. All wording (header, requirement fields, buttons, and input and output values) are in black to not exclude any colour-blind users, and the words have been bolded and displayed in a larger than usual font to help those with sight limitations. The buttons have been made larger as well to be easier to click. The requirement fields and buttons are kept simple and easy to understand as not to confuse anyone. Last of all, it is JavaScript accessible, meaning that interaction functions detailed in the script are available to users on any device (apart from the hover function which requires a mouse, however the calculator still works on devices that do not support hover functionality).

There are four golden rules in UI design, all of which I have implemented. The first rule, that places users in complete control of the interface, can be seen with the ‘reopen calculator’ button that allows the user to backtrack on their decision to leave the calculator without it becoming a hassle, and- as stated previously- the visual interface is simple and easy to navigate given its simple layout.

I mentioned quite a lot of this in the accessibility principles, but elements have been implemented to make it easier for users to interact with the webpage. This included larger buttons to make them easier to click, larger font to make it easier to read, and keeping the content simple and only including necessary functions. Colour was used sparingly, and only to add a level of contrast, from one feature to another, to the overall calculator from the rest of the webpage. Effective error messages that allow users to easily recover from mistakes have been implemented, and the input field does not automatically delete the users’ entered values. Lastly, jargon that people are more commonly aware of has been used instead of system-oriented items e.g., using ‘x’ to signify multiplication over ‘\*’.

Cognitive load has been reduced as a result of a lot of these UI elements, ensuring the ease of its usability. While the wording is in English, non-English users should not have a hard time using the app as it uses globally understood imagery and symbols to convey mathematical processes (‘x’ instead of ‘multiply’/’multiplication’). And last of all, the user interface remains consistent, with buttons and visuals staying the same regardless of webpage changes.

Interactive features:

The app that I decided to create was an interactive calculator. This included ten interactive features in total. The first two interactive features are input boxes that allow the user to insert a number value into each box. This was first implemented by using the form and input elements into the HTML, which allows a user to input a value, and is then targeted (via the input’s class name in the HTML) and attached to a new variable within the JavaScript via the querySelector(). A condition has been set to ensure that both inputted values need to be actual numbers that the calculator can use by ensuring that any value that turns up as a ‘NaN’ datatype (“not a number”) will trigger an error message to appear.

Following that, the next seven features are buttons, the first five being named “+”, “-“, “X”, “÷”, and “%”. These were first defined within the HTML element using separate class names, which were then identified and attached to new variables in the JavaScript, again using querySelector(). From there, each button needed a script to ensure that something would happen when interacting with them- in this case, by clicking the buttons with the mouse cursor. I accomplished this by adding event handlers to trigger a function when they are clicked. Their functions would vary depending on the button, where each one would trigger three actions: reading the numerical values inserted by the user, implementing a mathematical calculation to those inputted values depending on whether the specific button adds, subtracts, multiplies, divides, or modulates 2 values, and then displaying the result of those calculations.

The last three buttons were not for calculations, they only added interactive features to improve the usability of the calculator. The first button was the ‘clear’ button, which also used an event handler to create a trigger that, when clicked, would remove all values that were inputted by the user, and the result that was outputted by the calculator. It did so by converting the values into empty strings, and replaced the result output to one that did not include a sum result (a separate variable) of calculations.

Next was the ‘close’ button, designed to completely close the calculator and replace it with a message that would thank the user for using the calculator app. The script would first query select the close button, and the container of the body that contained the calculator, and assign them to new variables easily usable in the script. By using an event handler, the script -when the close button is clicked- will trigger a function that changes the calculator app with a thank you message using DOM manipulation. After it displays the message, it applies the styles to the message text, and then sets the ‘display’ style of the close button to ‘none’, hiding the button from the app user.

The last interactive feature allows the user to hover their mouse cursor over the buttons to indicate which button the user is on before committing to clicking onto it by using event handlers. The first event handler finds each item that belongs to the ‘button’ element, and then applies two events; the first ‘mouseenter’ event, when the cursor is actively hovering over any of the buttons, changes the styles; the other ‘mouseleave’ event simply reverts those styles to their original state once the cursor is no longer hovering over the button.