In this assignment, you will develop a calculator app using JavaScript

In completing this assignment, you will:

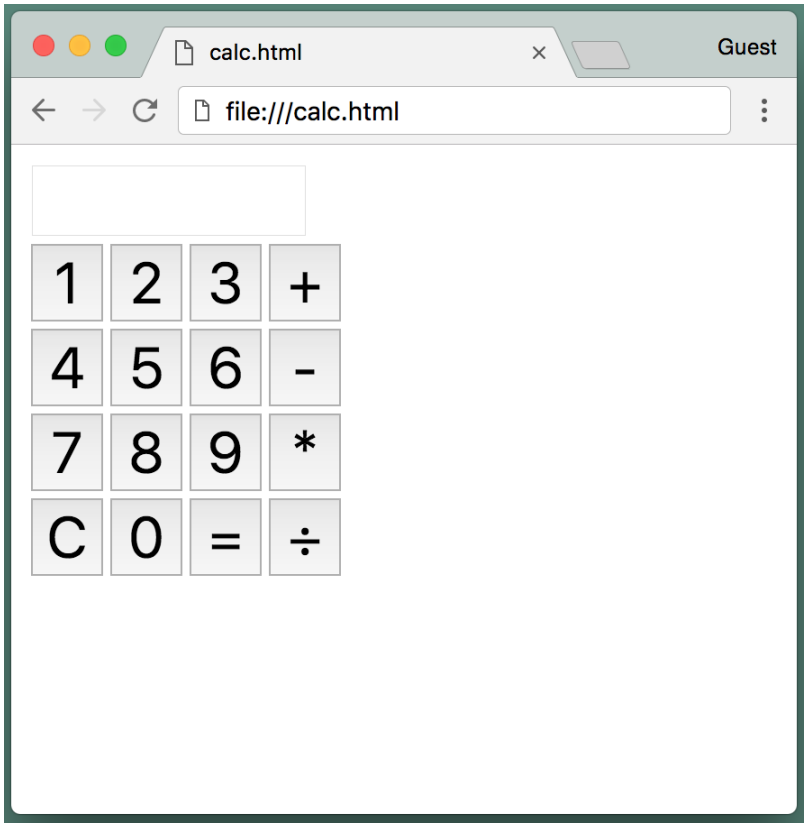
- Use the JavaScript Selectors to select HTML elements and modify their contents
- Define callback functions that are invoked as the result of user actions in the HTML page
- Assemble a JavaScript application consisting of multiple functions

Debugging/Error Note:

Do not change the existing IDs in the HTML document, but you are free to add HTML elements, such as Classes.

Background

We will provide you with the HTML that renders a simple calculator that looks like this:



In this assignment, you do not need to modify the appearance of the calculator, you only need to implement the functionality of entering values and

performing arithmetic operations using JavaScript in the web page and JavaScript Selectors to access the HTML elements.

Getting Started

Start by downloading the three files you will need for this assignment:

- Please save "calc.html" and "calc.js" to your computer from the classroom

The calc.html file is an HTML page that you can open in your browser and consists of the basic layout of the HTML elements that comprise the calculator. In particular, this page provides the HTML for:

- An input field at the top with the ID “display,” which is where the calculator’s output will be shown. This field is marked as “disabled” so that the user may not directly enter values into it.
- Buttons for the digits 0-9, with IDs “button0,” “button1,” etc.
- Buttons for the add, subtract, multiply, and divide operations, with IDs “addButton,” “subtractButton,” etc.
- Buttons with the IDs “equalsButton” and “clearButton”
- A span with the ID “output” that you can use for debugging and displaying other output, but is not part of the calculator function itself

At the top of the file is some CSS that modifies the appearance of the buttons and input field to make them a little bigger and easier to read. You may change this CSS styling if you prefer.

Last, at the bottom of calc.html is a `<script>` tag that includes “calc.js,” which is where you will implement all of the JavaScript code.

Activity

In calc.js, write the JavaScript code using Javascript to implement the calculator functionality. Your calculator should work as a “normal” calculator would be expected to operate, but here are the different use cases that your app needs to consider:

Case #1. Performing an operation on two numbers

Step	Action	Display (using example values)
1	The page is first loaded	(empty)
2	The user clicks a numbered button, e.g. “4”	4
3	The user clicks another numbered button, e.g. “3”	43
4	The user clicks an operator button, e.g. “+”	43
5	The user clicks a numbered button, e.g. “7”	7

6	The user clicks another numbered button, e.g. “1”	`71`
7	The user clicks the equals button. In this case, the app performs the most recent arithmetic operation on the two most recent numbers that were input	`114`

Other considerations:

- If the user chooses the divide operation and the result is not an integer, it should be displayed using floating point notation, e.g. “10” divided by “4” should produce “2.5”.
- If the user chooses the subtract operation and the result is negative, it should be displayed as a negative number, e.g. “5” minus “8” should produce “-3”. This includes subtracting from zero, too, of course.
- If the user attempts to divide by 0, the result should be shown as “Infinity”.

Case #2. Continuing an operation

Step	Action	Display (using example values)

1	Steps 1-7 for Case #1 above	`114`
2	The user clicks an operator button, e.g. “-”	`114`
3	The user clicks a numbered button, e.g. “5”	`5`
4	The user clicks another numbered button, e.g. “2”	`52`
5	The user clicks the equals button. In this case, the app performs the arithmetic operation on the result of the previous operation and the one that was most recently entered	`62`

Case #3. Starting a new operation

Step	Action	Display (using example values)
1	Steps 1-7 for Case #1 above	`114`
2	The user clicks a numbered button, e.g. “1”	`1`
3	The user clicks another numbered button, e.g. “0”	`10`
4	The user clicks an operator button, e.g. “*”	`10`
5	The user clicks a numbered button, e.g. “6”	`6`

6	The user clicks the equals button. In this case, the app performs the arithmetic operation on the two numbers that were input, and ignores the result of the previous operation	60

Case #4. Performing an operation on multiple numbers

Step	Action	Display (using example values)
1	The page is first loaded, or a prior operation is completed using the equals button	
2	The user clicks a numbered button, e.g. “1”	1
3	The user clicks another numbered button, e.g. “3”	13

4	The user clicks an operator button, e.g. “+”	13
5	The user clicks a numbered button, e.g. “7”	7
6	The user clicks an operator button, e.g. “-”. In this case, the app performs the arithmetic operation on the two numbers that were input, using the operator that was selected between entering them.	20
7	The user clicks another numbered button, e.g. “2”	2
8	The user clicks the equals button. In this case, the app performs the arithmetic operation on the result of the previous operation and the one that was most recently entered	18

After step 7, the user should of course be able to continue repeating steps 6 and 7 and performing (and seeing the result of) additional operations before selecting the equals button.

Case 5: Using the clear button

During or after any of the cases above, if the user clicks the clear button, then the app should reset itself back to the state in which the page was just loaded. It should not reload the page, of course, but rather should clear the display and “forget” the results of prior inputs or operations.

Case 6: Using the equals button

If the app is in the “reset” state – because the page has just been loaded, or because an operation was just completed, or because the user clicked the clear button – and the user enters one or more numbers and then clicks the equals button without first selecting an operator and entering another operand, the display should be the same and the equals button should be ignored. For instance, if the app is reset and the user clicks “2” and then “3” and then “=”, the display should still read “23” and that value should be used as normal for the next button click.

Likewise, if the app is in the reset state and the user enters some numbers, and then an operator, and then clicks the equals button without entering another operand, the display should be the same and the equals button should be ignored. For instance, if the app is reset and the user clicks “1” and then “5” and then “+” and then “=”, the display should still read “15” and that value and the “+” operator should be used as normal for the next button click.

However, if the user has just completed an operation using the equals button and then clicks the equals button again, the previous operation should be repeated using the result of the operation and the most recently entered operand. For instance, if the app is reset and the user enters “8” and then “+” and then “6” and then “=”, the display should read “14” as normal. If the user enters “=” again, the “+6” operation should be repeated and the display should now read “20”. If the user then enters “=” again, the “+6” operation should again be repeated and the display should read “26” and so on.

Case 7: Selecting multiple operators

If the app is in the “reset” state and the user enters some numbers, and then an operator, and then a different operator, the first operator should be ignored and the second operator should be used in the operation. For example, if the app is reset and the user enters “6” and then “+” and then “*” and then “2” and then “=”, the “+” operator should be ignored and the “*” operator should be used, so the display should read “12”.

What about…?

You may encounter other cases that are not addressed in this document, e.g. what to do when the result of an operation exceeds the largest number that JavaScript can represent, or other sequences of clicking buttons that do not follow the ones described above. You may handle those cases in any manner you choose (or ignore them entirely!) since they will not be considered for grading in this assignment.

One more important note:

Please do not change the IDs of any of the buttons or the “display” input field, as these IDs will be used by our tests during grading. You are free to add other attributes to the HTML elements, such as “class” attributes, but do not change the IDs.

DO NOT use any ES6 notation, as the autograder will not catch it. Additionally, do not use anything other than Class or ID selectors, such as $(.myClass) or $(#myID).

Likewise, please do not change either of the `<script>` tags in calc.html so that the grader can use the right .js files.

One more important note:

Review the past few lessons to see the syntax for selecting HTML elements and registering callback functions.

Keep in mind that JavaScript will perform concatenation on string variables, even if the values are numeric. So don't be surprised if you run into a situation in which '5' + 1 = '51'. You can use the JavaScript `Number()` function to convert a string variable to a numeric variable.

Before you submit

Please be sure that:

- You are using the calc.js file that we provided and have not changed either of the `<script>` tags in calc.html
- You have not changed the "id" attribute of any of the HTML elements in calc.html

- You have not created any additional files and all of your JavaScript code is in calc.js

Be sure to upload both **calc.html** and **calc.js** for grading.