Homework 2: JavaScript Calculator

Due date: January 31, 2022 at 11:59pm

This homework is the second in a series of three homework assignments in which you will build calculator applications. In this homework, you will implement the calculator functions using JavaScript running client-side in the web browser. (For the third homework, you will implement the calculator functions by sending web request to a Django application running server-side.)

For this assignment, you will use the RPN-calculator front end you created in the last assignment and add functionality by implementing the calculator behavior with JavaScript.

The learning goals for this assignment are to:

- Gain familiarity with client-side programming by using JavaScript to add functionality to a static front end through DOM manipulation and event dispatch.
- Ensure robustness of a small-scale application with some simple validation.

Specification

This section describes the behavior of the RPN calculator that you will implement.

- The calculator will use the front end that you wrote for the previous homework.
- The displayed output is always a single integer value, unless there is an error. This value is initially zero (0).
- The calculator does integer math (e.g., 8 ÷ 3 = 2).
- The calculator supports a maximum stack depth of three.
- One subtlety of calculator implementations is knowing, when a digit is pressed, whether you are starting to enter a new number or appending a digit to the number currently displayed. The rule is that after a math operation is performed (+, −, ×, or ÷), pressing a digit starts causes the result shown to be pushed (a “free push”) and a new number started. You must indicate which mode your calculator is in by changing the background color of the output area. (So, color a if clicking a digit continues the current number, and color b if causes a “free push”.)
- If the user attempts to (i) divide by zero, (ii) underflow the stack, or (iii) overflow the stack, you must output (i) "divide by zero", (ii) "stack underflow", or (iii) "stack overflow", respectively. In addition, you must set the background of the output area to some other color (neither a nor b).

Below are a few examples of the expected behavior of the calculator. Note that the stack below grows from the bottom up. Green indicates the free push. Red indicates an error.

Button Pressed	Init	2	4	↑	1	9	+	2	0	×	1	0	–
Stack	0	2	24	24 0	24 1	24 19	43	43 2	43 20	860	860 1	860 10	850
Output	0	2	24	0	1	19	43	2	20	860	1	10	850

Button Pressed	Init	6	8	↑	9	7	–	1	5	↑	4	÷	+
Stack	0	6	68	68 0	68 9	68 97	-29	-29 1	-29 15	-29 15 0	-29 15 4	-29 3	-26
Output	0	6	68	0	9	97	-29	1	15	0	4	3	-26

Button Pressed	Init	7	↑	8	↑	9	↑	7	+	1	↑	2	+
Stack	0	7	7 0	7 8	7 8 0	7 8 9	0	7	0	1	1 0	2	3
Output	0	7	0	8	0	9	OF	7	UF	1	0	2	3

Requirements

Your submission must also follow these requirements:

- Your submission must follow all of the requirements specified in the last homework assignment. Specifically, you must include the IDs on the HTML elements, as specified in Homework #1.
- You may not use any external libraries (e.g. jQuery, Bootstrap, etc.).
- You may not use the JavaScript `eval()` function in the implementation of your calculator.
- Your calculator must support numbers up to ±1,000,000. We will not test beyond that range.
- Your JavaScript may not crash at any user input—there must not be any errors reported by the browser JavaScript console.
- All your JavaScript functions must be defined in a file called `calculator.js`. Minimize the amount of JavaScript written directly in your HTML file.
- Cite all resources in a README.md file, using the same format as HW#1.

Implementation hints

- As discussed above, calculator is in one of two modes when a digit is pressed. Maintain a boolean variable to keep track of this mode. In this discussion, we'll call it "`entering`", which will be `true` when the new digit will be appended to the number currently displayed.

- Maintain the stack as a JavaScript array. You may maintain the "stack" as three integers, but JavaScript arrays have push and pop operations, so they are convenient for this homework. Regardless of your implementation, in this discussion, let's call the stack "`s`", let's say that numbers are pushed and popped from the bottom of the stack (and the end of the array), and let's call the number at the bottom of the stack "`bot`".

  So, in JavaScript code, you can create an empty stack with `s=[]`, push a value x onto the stack with `s.push(x)`, use `s[s.length-1]` to access "`bottom`", and pop off the stack with `s.pop()`. Note that `s.pop()` will return the value popped off the stack.

- When the calculator initializes:
  - `s = []`
  - `s.push(0)`
  - `entering = true`

- When a digit is pressed and `entering` is `true`:
  - `bottom` = `bottom` * 10 + `digit` (note that bottom is really `s[s.length-1]`)
  - Display as output the new value of `bottom`

- When the push operation (↑) is clicked:
  - Check to see that the stack is not full – if so, it's an overflow (see below)
  - `s.push(0)`
  - Display as output the value of `bottom` (so, display `0`)
  - `entering = true`

- When a math operation (+, −, × or ÷) is clicked:
  - Check to see that the stack size is at least two – if not, it's an underflow (see below)
  - Pop off the first two numbers from the stack
  - Compute the result of the operation – if divide by zero, see below
  - Push the result onto the stack
  - Display as output the result
  - `entering = false`

- When a digit is pressed and `entering` is `false`, you get a free push:
  - Check to see that the stack is not full[1] -- if so, it's an overflow
  - `s.push(0)`
  - `entering = true`
  - Then proceed as described above for digits when `entering` is `true`.
    - Multiply `top` by `10` and add the value of the digit (which just sets `bottom` to digit)
    - Display the value of `bottom`
- If you have an underflow, overflow, or divide by zero:
  - Display in the output the appropriate error message
  - Reinitialize the calculator, as described above:
    - `s = []`
    - `s.push(0)`
    - `entering = true`
- Note: Your calculator (as specified above) will not be able to support entering negative numbers, but it should be possible to use the negative numbers resulting from previous operations.
- Want more fun? Add an additional button (or buttons) to implement an additional operation(s), such as x^y (exponentiation), $^1/_x$ (inverting the value), or $^+/_-$ (inverting the sign).

Turning in your work

Your submission should be turned in via Git in a directory called **`hw2`** and should consist of an HTML file called `calculator.html`, a JavaScript file called `calculator.js`, and associated assets (CSS and images). Organize your files with the CSS and JavaScript files in their own folders as shown:

```
[YOUR-ANDREW-ID]/hw2/
  |-- calculator.html
  |-- css/
     |-- calculator.css
  |-- js/
     |-- calculator.js
  |-- README.md
```

Grading

You must run the grading script to get credit for this assignment. The grading script is (will be) available at https://grader.cmu-webapps.org. You may correct, resubmit, and regrade your homework as described in the syllabus.

[1] Note that the `entering` boolean variable is only set to `false` after a math operation and all our math operations use two values, so our calculator cannot overflow on the free push, unless you implement (additional) operations that use just one value from the stack (like invert value or invert sign).