Link web: Calculator - Basic Mode

Functional Specifications

Purpose and Scope

The purpose of this Web-Based Calculator is to provide a functional and reliable tool for basic arithmetic and standard operations, mirroring the functionality of the **Standard** mode of the Windows 11 calculator. It is a client-side web application designed to be simple and intuitive.

Supported Features

The calculator supports the following features, like the Windows 11 Standard mode:

Feature	Description	Function in Code
Number Input	Allows input of numbers (0-9)	appendNumber()
Basic Operations	Addition (+), Subtraction (-), Multiplication (×), Division (÷).	setOperator() calculate()
Equals (=)	Computes the finally result of the current expression.	equal()
Clear Entry (CE)	Clear the current input without clearing previous calculations.	clearEntry()
Clear (C)	Clears the entire calculation, resetting all state variables.	clearAll()
Backspace (⌫)	Deletes the last digit of the current input.	backspace()
Decimal Point (.)	Allows input of floating-point numbers.	appendDecimal()
Negate (±)	Negates the current input value.	negate()
Percentage (%)	Calculates the percentage of the current value based on the previous operand and operator.	percentage()
Square Root (√)	Calculates the square root of the current input.	squareRoot()
Square (x^2)	Squares the current input.	square()
Inverse (1/x)	Calculates the reciprocal (inverse) of the current input.	inverse()

Display	Show current and previous values.	updateDisplay()
History	Stores calculation history and allows loading previous results.	addToHistory(), loadFromHistory()
Error Handling	Displays specific messages for division by zero and square root of negative numbers.	handleError()
Keyboard Support	Accepts numerical and operator inputs via the keyboard.	document.addEventListener('keydown')

User Input, Operators, and Display Handling

Component	Handling Description
Number Input	Numbers are appended to the input string, replacing '0' if it's the only character. It respects a maximum length of **17 digits** (MAX_INPUT_LENGTH). The input resets after an operator or the equals button are clicked, or after a function button if the previous state was a function.
Operator Input	When an operator is clicked, the current input is set as currentValue, and a calculation is immediately performed if a previousValue and an operator are already present. The result becomes the new previousValue. The new operator is stored, and the expression display is updated. The state variable isClickedOperator is set to true.
Equals (=)	The equal() function uses the stored previousValue, operator, and the current input (as currentValue) to perform the final calculation. It also stores the operator and operand (lastOperator, lastOperand) to allow for continuous calculation repetition (e.g., $5 + 2 = 7 \rightarrow = 9 \rightarrow = 11$). The result is displayed, and the expression is finalized.
Display	The main display (display) shows the current input value. The secondary display (expressionDisplay) shows the ongoing formula (expression). Both displays use a dynamic font size adjustment (adjustFontSize) to ensure the number fits within the screen.
Error State	When a mathematical error occurs (e.g., divide by zero), the handleError() function is called. This sets the input to a descriptive error message, sets isErrorState to true, and disables all function, operator, and decimal buttons until the user presses **C, CE, ⌫,** and **any numbers**.

Assumptions

1. **Operator Precedence:** The calculator is implemented as a simple four-function calculator, executing operations immediately as they are entered. For example, $2 + 3 \times 4$ will be calculated as $(2 + 3) \times 4 = 20$, not $2 + (3 \times 4) = 14$.

2. **Number Format/Rounding:** Results are formatted into a string using the formatResult() function. If the result is a non-finite number, it displays **'Error'.** For numbers exceeding the maximum input length (MAX_INPUT_LENGTH = 17), they are converted to **exponential notation** using toExponential().
3. **Percentage Behavior:** The % key's behavior depends on the preceding operator, aligning with Windows Calculator Standard mode:
    - For **Addition** (+) and **Subtraction** (-), $X\ \{op\}\ Y\ \%\ \ =\ X\ \{op\}\ (X \times Y \div 100\ )$.
    - For **Multiplication** (x) and **Division** (÷), $X\ \{op\}\ \ Y\ \%\ =\ X\ \{op\}\ (Y \div 100)$.
    - If no operator is set or previousValue is null , the input is set to 0.

Non-Functional Specifications

Specification	Details
Performance	Operations execute synchronously and instantly. updateDisplay(), updateExpressionDisplay() and adjustFontSize() ensure quick and smooth UI updates, especially when input length changes.
Usability	The interface is designed to be clear with distinct buttons for numbers, operators, and functions. The history feature enhances usability by allowing users to recall past calculations. Error messages are simple and informative.
Cross-browser Compatibility	The calculator is built using pure HTML and CSS. Core logic is implemented with standard JavaScript, utilizing only document.getElementById and document.querySelectorAll for DOM manipulation. The code is fully compatible with **Chrome, Edge, Firefox, and Safari**.
Responsiveness	Font size adjustment is dynamic, supporting different screen widths by preventing text overflow.
Reliability and Maintainability	The state of the calculator is managed by clearly named global variables (currentValue, previousValue, operator, isErrorState, etc.), making the flow of logic easy to follow and debug. The use of separate functions for each operation (appendNumber, setOperator, calculate, etc.) promotes modularity.

Acceptance Criteria

Criterion	Fulfillment

Arithmetic operations return correct results.	Verified by the calculate() function and subsequent test cases.
Operator precedence is correctly applied.	**Not applicable.** The calculator is a simple sequential execution model (no standard precedence).
CE, C, and Backspace behave as in Windows Calculator.	**CE (clearEntry):** Clears current input, keeps calculation state. **C (clearAll):** Resets all state variables. **Backspace:** Removes the last digit of the current input.
Display updates accurately after each input.	updateDisplay() is called after every relevant user interaction, ensuring the current input and expression are displayed correctly.
Design remains stable across browsers and devices.	Uses standard web technologies and dynamic font sizing. However, if the display area is too small, it may become difficult to use the calculator due to its minimum layout size requirements.
Deployed version is publicly accessible and functional.	The project is uploaded to a public GitHub repository and deployed through Render to function as an accessible web application.

Testing Plan

The primary testing method will be **Manual Testing** using the web interface, focused on verifying the logic implemented in calculator.js.

Test Cases

Input	Expected Output	Actual Output	Result
$2 + 3 =$	5	5	Pass
$2.5 \times 4 =$	10	10	Pass
$10 \div 3 =$	3.3333333333333335	3.333333333333333	Pass
$10 \div 0 =$	$Cannot\ divide\ by\ zero$	$Cannot\ divide\ by\ zero$	Pass
$\sqrt{(-9)}$	$Invalid\ input$	$Invalid\ input$	Pass
$2 + 3 \times 4 =$	$20\ (\boldsymbol{Sequential\ execution})$	20	Pass
$10\ \% = (Default)$	0	0	Pass
$50 + 10\ \% =$	55	55	Pass
$50 \div 10\ \% =$	500	500	Pass
$123 \rightarrow \ \boxtimes$	12	12	Pass
$123 \rightarrow CE$	0	0	Pass
$123 \rightarrow C$	0	0	Pass
$9 \rightarrow \ x^2 \rightarrow =$	81	81	Pass
$9 \rightarrow \sqrt{x} \rightarrow =$	3	3	Pass
$2 \rightarrow \frac{1}{x} \rightarrow =$	0.5	0.5	Pass
$2 + 3 = \rightarrow =$	$8\ (\boldsymbol{Continuous\ operation})$	8	Pass

$1.234567890123456 \times 10 =$	12.34567890123456	12.34567890123456	Pass
$10 + 20 \rightarrow CE \rightarrow 5 \rightarrow =$	15	15	Pass
$3 + 6 = \rightarrow 5 \rightarrow =$	11 ($\boldsymbol{Continuous\ last\ equal}$)	11	Pass

---

Prompt Engineering (AI Assistance)

Claude AI - Structure Generation and Professional UI Design

Claude AI was used to generate the overall file structure and develop a visually professional and responsive interface for the calculator.

It focused on the UI/UX design, implementing clean layouts, dynamic font scaling, and cross-browser compatibility using pure HTML, CSS, and JavaScript.

Prompts used:

- **Prompt 1:** "Create a complete web-based calculator using raw HTML, CSS, and JavaScript that replicates the Windows 11 Basic Mode calculator." *(Enclosed: Required Features — Basic Mode Only table)*
  → Generated the initial project structure, defining all basic calculator functions and layout.
- **Prompt 2:** "Continue option, I want to have a view of the calculation history and delete buttons (all or element) to delete any element in history list, they appear on the right of the main calculator."
  → Added a calculation history panel and delete functionality.
- **Prompt 3:** "Make it more professional and beautiful."
  → Enhanced the aesthetic appeal, color harmony, and responsive design for a professional interface.

ChatGPT - Debugging and Logic Optimization

ChatGPT was primarily responsible for debugging, refining calculation logic, and improving the interaction experience.

It helped ensure the JavaScript logic matched real calculator behavior, especially the percentage operation and expression parsing.

Prompts used:

- **Prompt 1:** "How to debug JavaScript code embedded in HTML. I use VS Code to code and I run the HTML file in Edge."
  → Guided the setup of debugging tools and breakpoints for in-browser and VS Code inspection.
- **Prompt 2:** "How does the percent function work in Windows 11 calculator? Give me the JavaScript code for it."
  → Provided logic for percentage computation.
- **Prompt 3:** "Explain and give advice about this code." *(Enclosed: function step by step…)*
  → Offered step-by-step explanations of functions, variable scopes, and improvements for clarity and maintainability.

Google Gemini - Report Generation and Documentation

Google Gemini was used to automate documentation and generate structured project reports and README files.
It analyzed the provided JavaScript source code and followed a predefined documentation structure to produce complete technical content.

Prompts used:

- **Prompt 1:** "Write the report following the enclosed structure and fit the content with the .js file." *(Enclosed: calculator.js and documentation sections)*
  → Generated detailed sections explaining functionality, testing plans, and non-functional requirements.
- **Prompt 2:** "Write a README.md to introduce my project and its features. Make it more professional. Write it so I can copy it directly into a .md file."
  → Produced a polished and well-organized README file suitable for public GitHub hosting.

Reflection on AI Usage

Throughout the development process, I learned how to effectively collaborate with multiple AI tools to complete a full software project—from initial design to deployment. Using **Claude AI**, I understood how to structure a project and design a professional-looking user interface based on prompts and structured requirements. **ChatGPT** helped me deeply understand JavaScript logic, debugging methods, and browser-based compatibility, which improved my coding precision and confidence when solving logical or functional issues. Meanwhile, **Google Gemini** supported me in documenting and structuring the final report, ensuring the content was clear, organized, and professional.