Simple Calculator

Abstract:

This project implements a Simple Calculator using core web technologies: HTML, CSS, and JavaScript. The calculator provides an elegant and modern aesthetic with blurred background effects and a smooth user interface. The design supports a light/dark mode toggle, allowing users to switch between themes for a personalized experience.

The calculator supports basic arithmetic operations, including addition, subtraction, multiplication, and division, with intuitive button layout and responsive design. The JavaScript functionality handles dynamic input, displaying mathematical expressions and results in real-time. In addition, keyboard input is supported for convenience, and the calculator includes error handling for invalid expressions.

Introduction:

This project focuses on building a Simple Calculator using HTML, CSS, and JavaScript, showcasing the integration of these technologies to create a functional and visually appealing web application.

The calculator performs basic arithmetic operations like addition, subtraction, multiplication, and division. It features design with a frosted glass effect, along with a light/dark mode toggle for improved user experience.

HTML structures the layout, CSS handles styling , and JavaScript manages the calculator's functionality, including button interactions, input display, calculations, and error handling. The project demonstrates core web design principles and offers a foundation for future enhancements.

Objective:

The primary objectives of this project are:

1. **To create a functional and interactive calculator: Build a basic calculator that supports arithmetic operations such as addition, subtraction, multiplication, and division. The calculator should allow users to perform real-time calculations with intuitive input and output handling.**

2. **To integrate interactivity with JavaScript: Leverage JavaScript for managing dynamic inputs, handling button clicks, processing mathematical expressions, and displaying results. This includes supporting keyboard input and error handling to ensure smooth operation.**

3. **To demonstrate a responsive layout: Ensure the calculator is responsive, adapting seamlessly to different screen sizes and devices, providing an optimal user experience on both desktops and mobile devices.**

Project Design:

The design of the Simple Calculator focuses on creating an intuitive, aesthetically pleasing, and functional user interface. The project design is divided into three main components: structure, styling, and functionality.

1. Structure

The structure of the calculator is defined using HTML and is organized into the following sections:

- **Theme Toggle Button: Located at the top-right of the page, this button allows users to switch between light and dark modes. It is a simple button with a sun/moon icon that toggles the appearance of the entire page.**
- **Calculator Container: The main calculator container is wrapped inside a div element with the class .calculator. This container holds all the visual components, including the display and buttons.**
- **Display Section: The display area consists of two parts:**
    - **Expression Display: Shows the mathematical expression being entered.**
    - **Result Display: Shows the result of the current expression or the last calculated result.**
- **Buttons Section: The calculator buttons are organized in a grid layout using grid-template-columns, which defines a 4x5 grid. Buttons are added for digits (0-9), operators (+, -, *, /), and special buttons for clearing the display (Ac), parentheses ((,)), and the equals sign (=) for calculating the result.**

2. Styling

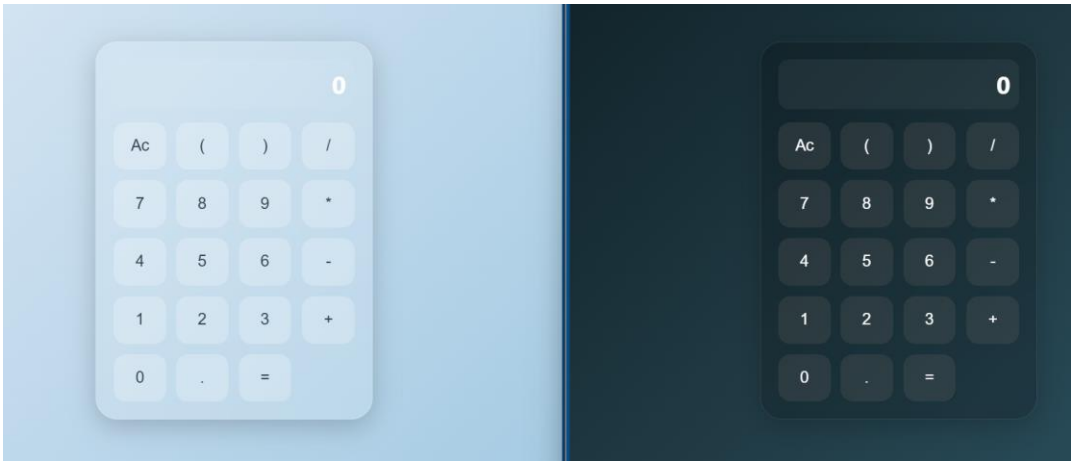
The calculator uses CSS for layout and design, focusing on the following visual aspects:

- **Light/Dark Mode: The design supports both light and dark modes. The light theme uses light gradient background colors, while the dark theme uses darker gradient colors. This is controlled by toggling between body.light and body.dark classes.**
- **Responsive Grid Layout: The calculator's buttons are arranged in a 4x5 grid, which adjusts seamlessly for various screen sizes, ensuring that the design remains user-friendly on both mobile and desktop devices.**
- **Hover Effects: Buttons have a subtle hover effect achieved through transition properties, which change the background color and make the interface more interactive.**
- **Typography and Spacing: The use of modern fonts such as 'Segoe UI', along with careful control over padding, margins, and font sizes, ensures readability and a clean layout.**

3. Functionality

The functionality of the calculator is powered by JavaScript, providing interactivity and dynamic behavior:

- **Input Handling: When a user clicks a button, the corresponding value is appended to the expression, which is displayed in the resultDisplay area. The input also supports keyboard input for convenience.**
- **Expression Calculation: When the equals button (=) is pressed, the eval() function is used to evaluate the expression entered by the user. The result is then displayed in the resultDisplay area. If an error occurs (e.g., invalid syntax), the display shows "Error."**
- **Clear Display: The Ac (clear) button resets the expression to an empty string and clears the display.**
- **Theme Toggle: The button with the id="toggleTheme" listens for a click event to switch between light and dark modes. The body class is toggled between light and dark to apply the appropriate theme.**
- **Keyboard Support: The calculator listens for key presses and responds to numbers, operators, parentheses, the enter key for calculation, and backspace for deleting the last character.**



Technologies Used:

This project leverages three fundamental web technologies: HTML, CSS, and JavaScript. These technologies work together to create a functional, visually appealing, and interactive web application.

1. HTML (HyperText Markup Language)

HTML is the backbone of the calculator's structure, defining the content and layout. It is used to:

- **Structure the overall layout of the calculator, including the display section and buttons.**
- **Organize the HTML elements such as div, button, and span to provide an accessible interface.**
- **Facilitate the interaction between the user and the application by defining the buttons for mathematical operations and numeric inputs.**

Key Features:

- **The div tags are used to create container sections, including the display and buttons.**

- **The button tags handle user input, triggering events when clicked.**
- **The id and class attributes are used for targeting and styling specific elements.**

2. CSS (Cascading Style Sheets)

CSS is used to style the calculator and apply the glassmorphism effect, making the user interface modern and visually appealing. The key aspects of the design handled by CSS include:

- **Light/Dark Mode: CSS classes (light and dark) allow the user to toggle between two themes. Background gradients, button colors, and other styling properties change dynamically when the theme is switched.**
- **Responsive Design: A grid layout is used for the buttons, ensuring the calculator adapts to different screen sizes and devices. The design is flexible and works across both mobile and desktop platforms.**
- **Button Styling: Buttons are styled with rounded corners, subtle hover effects, and a clean, minimalistic look that enhances the user experience.**

Key Features:

- **Use of grid layout for arranging buttons.**
- **Application of backdrop-filter and border-radius for a modern look.**
- **Styling of the theme toggle button and responsive layout adjustments.**

3. JavaScript

JavaScript provides the core functionality of the calculator, enabling dynamic behavior and interactivity. It is responsible for:

- **Input Handling: Capturing user input from both the on-screen buttons and the keyboard. This allows the user to input numbers, operators, and other symbols needed for the calculation.**
- **Expression Evaluation: When the user presses the equals button (=), JavaScript evaluates the mathematical expression using the eval() function and displays the result.**
- **Clear Functionality: The clear button (Ac) resets the calculator and clears the current expression and result.**
- **Error Handling: JavaScript handles invalid expressions and displays an "Error" message if the evaluation fails.**
- **Keyboard Support: The calculator supports keyboard input, enabling users to enter numbers and operators via the keyboard.**
- **Theme Toggle: JavaScript listens for the click event on the theme toggle button to switch between light and dark modes by toggling appropriate CSS classes.**

Key Features:

- **Dynamically updates the display based on user input.**
- **Handles the core calculator logic (evaluating the expression).**

- **Provides keyboard event listeners for ease of use.**
- **Implements error handling to display a message when the expression is invalid.**

4. Other Technologies and Tools

- **Web Browser: The project is intended to run on modern web browsers, including Chrome, Firefox, and Edge, ensuring cross-browser compatibility.**
- **Text Editor: Code was written and edited using a text editor such as Visual Studio Code (VS Code), which provides a user-friendly environment for coding with features like syntax highlighting and live preview.**

Implementation:

This section explains the step-by-step implementation process for building the Simple Calculator using HTML, CSS, and JavaScript. It covers the core functionalities, layout design, and integration of different components.

1. HTML Structure

The HTML structure is designed to create the basic layout of the calculator, which includes the display area and buttons. The structure is as follows:

- **<button> Elements: Buttons are created for numeric input (0-9), mathematical operators (+, -, *, /), parentheses ((,)), and other functional keys like "Clear" (Ac) and "Equals" (=).**
- **Display Section: The display area is divided into two parts:**
    - **Expression Display (#expressionDisplay): This section shows the current input or ongoing mathematical expression.**
    - **Result Display (#resultDisplay): This section shows the final result or the most recent calculation. It is initially set to "0" and updated as users enter input or calculate results.**

```
<div class="calculator">
  <div class="display-section">
    <div class="expression" id="expressionDisplay"></div>
    <div class="result" id="resultDisplay">0</div>
  </div>
  <div class="buttons">
    <button onclick="clearDisplay()">Ac</button>
    <button onclick="appendValue('(')">(</button>
    <button onclick="appendValue(')')">)</button>
    <button onclick="appendValue('/')">/</button>
    <!-- Additional buttons for digits and operations -->
  </div>
</div>
```

2. CSS Styling

The CSS part handles the design and layout, focusing on visual appeal and responsiveness. The key styles include:

- **Glassmorphism Effect: The use of backdrop-filter: blur(20px) creates the frosted glass effect for the calculator's background.**
- **Grid Layout for Buttons: The buttons are arranged using a CSS grid, ensuring they are evenly spaced and responsive.**
- **Light/Dark Mode: The body element's class is toggled between light and dark, dynamically changing the background gradients and button colors.**

The body class switching between light and dark mode is controlled by the JavaScript function tied to the theme toggle button.

```css
.calculator {
  width: 320px;
  padding: 20px;
  border-radius: 25px;
  background: rgba(255, 255, 255, 0.15);
  backdrop-filter: blur(20px);
}

body.light {
  background: linear-gradient(135deg, #dbe8f4, #a2c9e2);
}

body.dark {
  background: linear-gradient(135deg, #0f2027, #203a43, #2c5364);
}
```

3. JavaScript Functionality

The core functionality of the calculator is powered by JavaScript, which handles the input processing, expression evaluation, and result display. Here's how each part is implemented:

- **Input Handling: When a user clicks on a button, the corresponding value is added to the expression string. If the user presses the = button, the calculateResult() function is called, which evaluates the expression using the eval() function and displays the result.**

```
function appendValue(val) {
  if (finalResultShown) {
    // Start a new expression after showing the result
    expression = resultDisplay.innerText;
    finalResultShown = false;
  }
  expression += val;
  resultDisplay.innerText = expression;
}
```

- **Clear Function: The "Clear" button (Ac) resets the expression and result display to "0".**

```
function clearDisplay() {
  expression = "";
  resultDisplay.innerText = "0";
}
```

- **Calculation: The calculation is triggered when the equals button (=) is clicked. The eval() function is used to evaluate the mathematical expression. In case of an invalid expression, it catches the error and displays "Error".**

```
function calculateResult() {
  try {
    const result = eval(expression);
    resultDisplay.innerText = formatNumber(result);
    finalResultShown = true;
  } catch (e) {
    resultDisplay.innerText = "Error";
  }
}
```

- **Keyboard Support: The calculator listens for keyboard input, allowing users to type numbers and operators. The "Enter" key triggers the calculation, and "Backspace"**

```
document.addEventListener("keydown", (event) => {
  const key = event.key;
  if (!isNaN(key) || "+-*/().".includes(key)) {
    appendValue(key);
  } else if (key === "Enter") {
    event.preventDefault();
    calculateResult();
  } else if (key === "Backspace") {
    expression = expression.slice(0, -1);
    resultDisplay.innerText = expression || "0";
  }
});
```

- **Theme Toggle: The toggleTheme button changes the appearance between light and dark mode. When clicked, it toggles the classes light and dark on the body element and updates the icon.**

```
document.getElementById("toggleTheme").addEventListener("click", () => {
  document.body.classList.toggle("dark");
  document.body.classList.toggle("light");
  document.getElementById("toggleTheme").textContent = document.body.classList.contains("dark") ?
});
```

4. Error Handling

The calculator gracefully handles errors by displaying "Error" if the user enters an invalid expression or an operation that can't be calculated. This ensures that the application remains user-friendly and doesn't break during usage.

5. Responsive Design

The design is made responsive using CSS grid for the buttons, ensuring that the calculator layout adjusts correctly for various screen sizes. The calculator maintains usability across mobile, tablet, and desktop devices.

Testing:

Basic Functionality:

- **Ensure that all buttons (numbers, operators, clear, and equals) work as expected.**
- **Check that the calculator performs basic arithmetic operations (addition, subtraction, multiplication, and division).**
- **Verify that the result displays correctly after pressing =.**

Clear Function:

- **Test the Ac (clear) button to ensure it resets the input and result to the default state.**

Theme Toggle:

- **Click the theme toggle button (🌙 / 🌞) and confirm that the calculator switches between light and dark modes.**
- **Ensure that the background, button colors, and other styling elements change appropriately.**

Keyboard Input:

- **Try entering numbers and operators using your keyboard. Ensure that the calculator reacts correctly to key presses (e.g., typing 5, +, 3, and then pressing Enter should give the correct result).**
- **Test the backspace functionality by pressing the Backspace key.**

Error Handling:

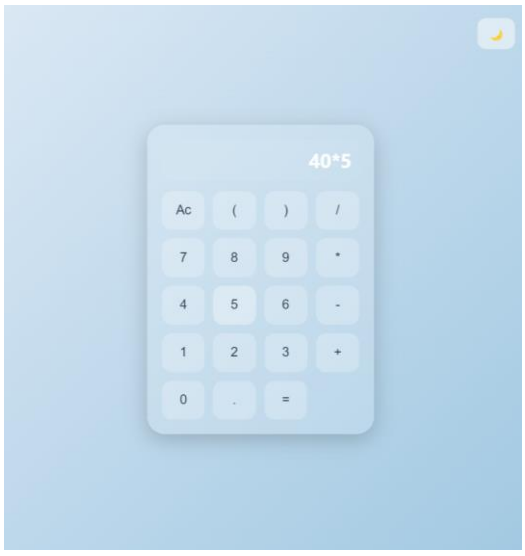
- **Try entering invalid expressions (e.g., 5 + or 7/0) to ensure that the calculator displays an "Error" message.**

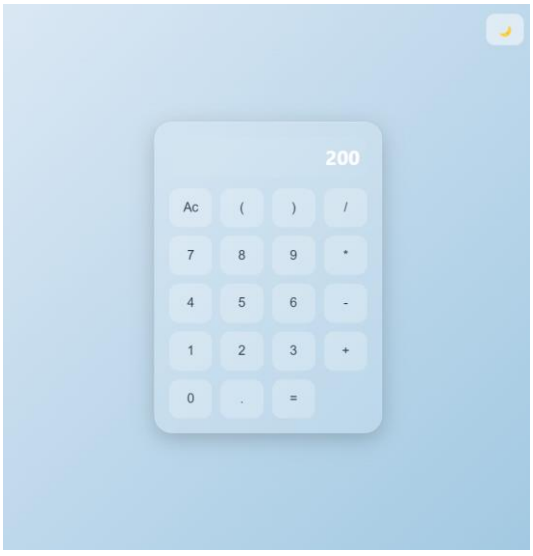
Responsive Layout:

- **Test the calculator on different screen sizes (mobile, tablet, desktop) to ensure the layout remains functional and visually appealing.**

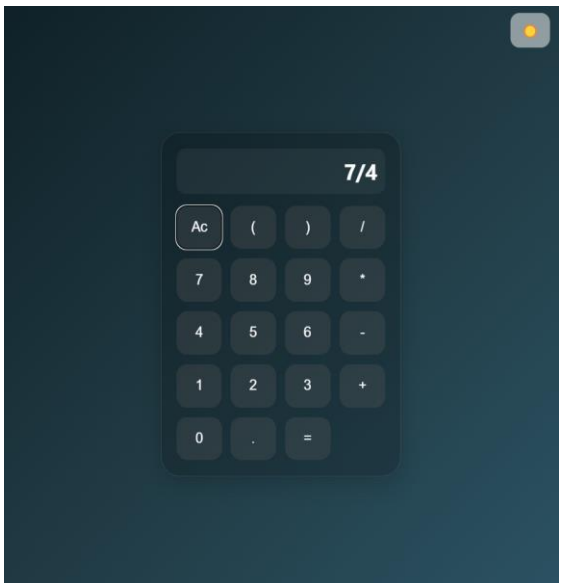
Execution Screenshots:

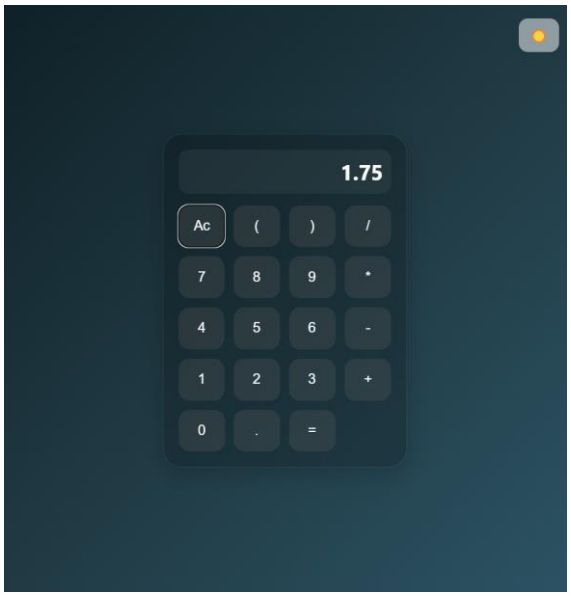
Light Theme:





Dark Theme:





Conclusion:

The Simple Calculator using HTML, CSS, and JavaScript project successfully implements a user-friendly, interactive calculator. The project achieves the following objectives:

- **Responsive and Functional Design: The calculator provides a clean, modern interface that adapts to various screen sizes, making it suitable for both mobile and desktop users.**
- **Light and Dark Mode: The inclusion of light and dark themes enhances the user experience by offering visual flexibility, which can be toggled easily.**
- **Core Calculator Features: All essential calculator functionalities, including numeric input, arithmetic operations, result calculation, and error handling, have been effectively implemented.**
- **Keyboard Support: The integration of keyboard input allows users to perform calculations using both mouse and keyboard, improving accessibility and convenience.**
- **User Experience: The design is not only visually appealing but also intuitive, with clear feedback for user actions, such as displaying the current expression and result.**

This project showcases the seamless integration of HTML for structure, CSS for design, and JavaScript for functionality, making it a comprehensive demonstration of web development skills. It serves as an excellent foundation for future improvements, such as adding more advanced mathematical operations or incorporating a history feature for previous calculations.

In conclusion, the project successfully meets the requirements and objectives, providing a fully functional, visually engaging, and interactive calculator that can be used on a variety of devices.

References:

References for Calculator made using Html JavaScript and CSS.

1. **WebDevSimplified. (2024).** ***How to Build a Simple Calculator with HTML, CSS, and JavaScript.*** **WebDevSimplified Blog.**
2. **MDN Web Docs. (2024).** ***CSS Grid Layout.*** **MDN Web Docs.**
3. **Stack Overflow. (2024).** ***Theme Toggle Button in JavaScript.*** **Stack Overflow Discussion.**
4. **FreeCodeCamp. (2023).** ***JavaScript for Beginners: Understanding Functions and Loops.*** **FreeCodeCamp.**
5. **JavaScript.info. (2024).** ***Introduction to JavaScript.*** **JavaScript.info.**

SOURCE CODE:

index.html styles.css script.js

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Calculator</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body class="light">
  <button id="toggleTheme">🌙</button>
  <div class="calculator">
    <div class="display-section">
      <div class="expression" id="expressionDisplay"></div>
      <div class="result" id="resultDisplay">0</div>
    </div>
    <div class="buttons">
      <button onclick="clearDisplay()">Ac</button>
      <button onclick="appendValue('(')">(</button>
      <button onclick="appendValue(')')">)</button>
      <button onclick="appendValue('/')">/</button>
      <button onclick="appendValue('7')">7</button>
      <button onclick="appendValue('8')">8</button>
      <button onclick="appendValue('9')">9</button>
      <button onclick="appendValue('*')">*</button>
      <button onclick="appendValue('4')">4</button>
      <button onclick="appendValue('5')">5</button>
      <button onclick="appendValue('6')">6</button>
      <button onclick="appendValue('-')">-</button>
      <button onclick="appendValue('1')">1</button>
      <button onclick="appendValue('2')">2</button>
      <button onclick="appendValue('3')">3</button>
      <button onclick="appendValue('+')">+</button>
      <button onclick="appendValue('0')">0</button>
      <button onclick="appendValue('.')">.</button>
      <button class="equal" onclick="calculateResult()">=</button>
    </div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

index.html styles.css script.js Calculator

```javascript
let expression = "";
let finalResultShown = false;

const resultDisplay = document.getElementById("resultDisplay");

function appendValue(val) {
  if (finalResultShown) {
    expression = resultDisplay.innerText;
    finalResultShown = false;
  }
  expression += val;
  resultDisplay.innerText = expression;
}

function clearDisplay() {
  expression = "";
  resultDisplay.innerText = "0";
}

function calculateResult() {
  try {
    const result = eval(expression);
    resultDisplay.innerText = formatNumber(result);
    finalResultShown = true;
  } catch (e) {
    resultDisplay.innerText = "Error";
  }
}

function formatNumber(num) {
  return Number(num).toLocaleString("en-US");
}

document.getElementById("toggleTheme").addEventListener("click", () => {
  document.body.classList.toggle("dark");
  document.body.classList.toggle("light");
  document.getElementById("toggleTheme").textContent = document.body.classList.contains("dark") ? "☀️" : "🌙"
});

document.addEventListener("keydown", (event) => {
  const key = event.key;
  if (!isNaN(key) || "+-*/().".includes(key)) {
    appendValue(key);
  } else if (key === "Enter") {
    event.preventDefault();
    calculateResult();
  } else if (key === "Backspace") {
    expression = expression.slice(0, -1);
    resultDisplay.innerText = expression || "0";
  }
});
```

index.html styles.css script.js

```css
* {
  box-sizing: border-box;
}
body {
  font-family: 'Segoe UI', sans-serif;
  margin: 0;
  padding: 0;
  height: 100vh;
  display: flex;
  align-items: center;
  justify-content: center;
  transition: background 0.3s ease;
}
body.light {
  background: linear-gradient(135deg, #dbe8f4, #a2c9e2);
}
body.light .buttons button {
  color: #2c3e50;
}
body.dark {
  background: linear-gradient(135deg, #0f2027, #203a43, #2c5364);
}
#toggleTheme {
  position: absolute;
  top: 20px;
  right: 20px;
  padding: 10px 14px;
  font-size: 18px;
  border: none;
  background: rgba(255, 255, 255, 0.50);
  border-radius: 12px;
  cursor: pointer;
  backdrop-filter: blur(10px);
  transition: 0.3s;
}
.calculator {
  width: 320px;
  padding: 20px;
  border-radius: 25px;
  background: rgba(255, 255, 255, 0.15);
  backdrop-filter: blur(20px);
  box-shadow: 0 8px 30px rgba(0, 0, 0, 0.2);
  border: 1px solid rgba(255, 255, 255, 0.25);
  display: flex;
  flex-direction: column;
  gap: 15px;
  transition: 0.3s;
}
body.dark .calculator {
  background: rgba(0, 0, 0, 0.2);
  border: 1px solid rgba(255, 255, 255, 0.1);
}
.display-section {
  display: flex;
  flex-direction: column;
  align-items: flex-end;
  padding: 10px;
  border-radius: 12px;
  color: white;
  background: rgba(255, 255, 255, 0.05);
}
.expression {
  font-size: 16px;
  opacity: 0.7;
}
.result {
  font-size: 28px;
  font-weight: bold;
}
.buttons {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
  gap: 12px;
}
.buttons button {
  height: 55px;
  font-size: 18px;
  border: none;
  border-radius: 14px;
  color: white;
  background: rgba(255, 255, 255, 0.1);
  backdrop-filter: blur(5px);
  cursor: pointer;
  transition: all 0.2s ease-in-out;
}
.buttons button:hover {
  background: rgba(255, 255, 255, 0.25);
}
.equal {
  background-color: #007BFF;
}
.equal:hover {
  background-color: #0056b3;
}
```