**Erstellung eines leeren Arrays für die Historie von Berechnungen und Ergebnissen:**

Ein leeres Array namens **history** wird erstellt, um die Historie der Berechnungen und ihrer Ergebnisse zu speichern.

**Verwendung einer Flagge operatorAdded:**

Die Variable **operatorAdded** wird verwendet, um festzuhalten, ob ein Operator zu einer aktuellen Berechnung hinzugefügt wurde. Sie wird später verwendet, um sicherzustellen, dass nicht mehrere aufeinanderfolgende Operatoren hinzugefügt werden.

**Funktion appendOperation(operation):**

Diese Funktion fügt Zahlen, Operatoren oder andere Zeichen zur aktuellen Berechnung im Anzeigebereich hinzu.

Es überprüft, ob die übergebene Operation ein mathematischer Operator ist und verhindert das Hinzufügen mehrerer aufeinanderfolgender Operatoren.

Wenn die Operation kein Operator ist, wird die **operatorAdded**-Flagge zurückgesetzt.

**Funktion calculate():**

Diese Funktion wird aufgerufen, wenn auf das Gleichheitszeichen (=) geklickt wird.

Sie validiert die Eingabe, führt die Berechnung durch und zeigt das Ergebnis an.

Wenn die Eingabe ungültige Zeichen enthält oder die Berechnung fehlschlägt, wird 'Fehler' angezeigt.

Die Berechnung wird mithilfe von **eval** durchgeführt (außer bei Exponenten, die gesondert behandelt werden).

**Funktion displayHistory():**

Diese Funktion zeigt die Berechnungshistorie an, indem sie den Bereich 'historyArea' mit den vergangenen Berechnungen und Ergebnissen aktualisiert.

**Funktion ClearHistory():**

Diese Funktion löscht die Berechnungshistorie, indem sie das **history**-Array leert und die Anzeige aktualisiert.

Sie bestätigt die Löschung der Historie durch eine eingehende Nachricht und löscht diese Nachricht nach einigen Sekunden.

**Funktion deletelast():**

Diese Funktion entfernt das letzte Zeichen aus der aktuellen Berechnung im Anzeigebereich.

Sie überprüft, ob das gelöschte Zeichen ein Operator war und aktualisiert **operatorAdded** entsprechend.

**Funktion clearDisplay():**

Diese Funktion löscht den gesamten Inhalt im Anzeigebereich (resultArea).

Sie setzt auch die **operatorAdded**-Flag zurück und zeigt eine Löschbestätigung an.

**Funktion calculatePercentage():**

Diese Funktion berechnet den Prozentsatz der aktuellen Zahl im Anzeigebereich, indem sie das Ergebnis durch 100 teilt und den Prozentsatz anzeigt.

Bei Fehlern wird der Benutzer darüber benachrichtigt.

**Funktion theNonFunctionalButton():**

Diese Funktion stellt eine Art Platzhalter für mögliche zukünftige Funktionen dar. Wenn auf die Schaltfläche geklickt wird, wird dem Benutzer eine Nachricht "irgendwann" angezeigt, die nach 3 Sekunden wieder verschwindet.

function:

Definiert ein grundlegendes Konzept, das ermöglicht einen Block von Code zu definieren und wiederzuverwenden. Sind die wichtigsten Bausteine in JavaScript und spielen eine entscheidende Rolle bei der Strukturierung und Organisation von Code.

const:

Ein Schlüsselwort, um Variablen zu deklarieren. Variablen die mit const deklariert. Sind sogenannte konstante Variablen, was bedeutet das nach der Initialisierung nicht mehr verändert werden können.

container:

Ist in JavaScript eine Variable, ist jedoch nicht spezifisch definiert, sondern gewählt. Es greift auf ein Element in der HTML-DOM zu normalerweise mit dem Ziel auf dieses Element zuzugreifen und zu manipulieren.

***Genauere Erklärung Regex:***

Der Regex-Ausdruck /[^0-9()+\-\*\/.^%]/ besteht aus den folgenden Elementen:

/ und / markieren den Anfang und das Ende des Regex-Ausdrucks.

[^0-9()+\-\*\/.^%] ist ein Negationsoperator, der alle Zeichen außer den folgenden ausschließt:

Zahlen (0-9)

Klammern ()

Operatoren (+, -, \*, /, ., ^, %)

***Erklärung des Beispiels:***

Wenn der Benutzer die Eingabe "1 + 2 \* 3" in das Anzeigefeld eingibt, wird der Regex-Ausdruck wie folgt ausgewertet:

[^0-9()+\-\*\/.^%] wird angewendet auf "1 + 2 \* 3".

Alle Zeichen außer den Zahlen, Klammern, Operatoren und Sonderzeichen werden entfernt.

Das Ergebnis ist "1+2\*3".

***Genauere Erklärung zur calculate() Funktion:***

Erklärung des Beispiels:

Wenn der Benutzer die Eingabe "1 + 2 \* 3" in das Anzeigefeld eingibt, wird der Regex-Ausdruck wie folgt ausgewertet:

[^0-9()+\-\*\/.^%] wird angewendet auf "1 + 2 \* 3".

Alle Zeichen außer den Zahlen, Klammern, Operatoren und Sonderzeichen werden entfernt.

Das Ergebnis ist "1+2\*3".

------------------------------------------------------------------------------------------------------------------------------------------------------------------

Zeile 1: Die Funktion wird als calculate() deklariert.

Zeilen 2-3: Die Variablen container und calculation werden initialisiert. container verweist auf das Anzeigefeld und calculation enthält den aktuellen Ausdruck.

Zeile 4: Die Funktion /[^0-9()+\-\*\/.^%]/.test() wird verwendet, um zu prüfen, ob der Ausdruck ungültige Zeichen enthält. Wenn dies der Fall ist, wird "Fehler" angezeigt und die Funktion wird beendet.

Zeilen 6-12: Wenn der Ausdruck gültig ist, wird die Berechnung durchgeführt. Wenn der Ausdruck einen Exponenten enthält, wird dieser separat berechnet. Andernfalls wird der Ausdruck an eval() übergeben, um die Berechnung durchzuführen.

Zeile 13: Das Ergebnis der Berechnung wird dann im Anzeigefeld angezeigt.

Zeilen 14-16: Die Berechnung und das Ergebnis werden in der Historie gespeichert.

***Was ist, wenn ein Exponent sich in der Berechnung findet?***

Der Bereich if (calculation.includes("^")) wird ausgeführt, wenn die Eingabe des Benutzers ein ^-Zeichen enthält. Das ^-Zeichen ist der Operator für die Potenzierung.

Wenn das ^-Zeichen vorhanden ist, wird der Ausdruck in zwei Teile getrennt. Der erste Teil ist die Basis und der zweite Teil ist der Exponent.

Die Basis wird dann als String in parseFloat() übergeben, um sie in eine Zahl zu konvertieren. Der Exponent wird ebenfalls als String in parseFloat() übergeben, um ihn in eine Zahl zu konvertieren.

Wenn die Basis und der Exponent gültig sind, wird das Ergebnis der Berechnung mit Math.pow() berechnet.

***Genaure Erklärung der einzelnen Zeilen***

Zeile 1: Der Bereich wird mit der Bedingung calculation.includes("^") ausgeführt. Diese Bedingung überprüft, ob die Eingabe des Benutzers ein ^-Zeichen enthält.

Zeile 2: Wenn die Bedingung erfüllt ist, wird der Ausdruck in zwei Teile getrennt. Der erste Teil ist die Basis und der zweite Teil ist der Exponent.

Zeile 3: Die Basis wird als String in parseFloat() übergeben, um sie in eine Zahl zu konvertieren.

Zeile 4: Der Exponent wird ebenfalls als String in parseFloat() übergeben, um ihn in eine Zahl zu konvertieren.

Zeile 5: Wenn die Basis und der Exponent gültig sind, wird das Ergebnis der Berechnung mit Math.pow() berechnet.

Zeile 6: Wenn die Basis oder der Exponent ungültig sind, wird "Fehler" angezeigt und die Funktion wird beendet.

calculation.includes("^") ist wahr, da die Eingabe ein ^-Zeichen enthält.

calculation.split() wird angewendet auf "2^3".

Das Ergebnis ist ["2", "3"].

parseFloat() wird angewendet auf "2" und "3".

Das Ergebnis ist 2 und 3.

Math.pow() wird angewendet auf 2 und 3.

Das Ergebnis ist 8.

***Genauere Erklärung eval():***

eval() ist eine mächtige Funktion, die JavaScript-Code ausführt, der als String übergeben wird. Das macht sie zu einem vielseitigen Werkzeug, das für eine Vielzahl von Aufgaben verwendet werden kann. Im Fall des gerade eingegebenen JavaScript-Codes wird eval() verwendet, um die Berechnung durchzuführen. Der aktuelle Berechnungsstring wird an eval() übergeben und die Funktion führt den Code aus, um das Ergebnis zu berechnen.

eval() ist jedoch auch eine unsichere Funktion. Sie kann verwendet werden, um JavaScript-Code aus unerwarteten Quellen auszuführen. Dies kann zu Sicherheitslücken führen. Um die Sicherheit von eval() zu erhöhen, sollte man nur Code aus vertrauenswürdigen Quellen ausführen. Außerdem sollte man eval() nur verwenden, wenn dies unbedingt notwendig ist.

Im Fall der calculate()-Funktion wird eval() verwendet, um den aktuellen Ausdruck auszuführen. Dies ist eine einfache Möglichkeit, die Berechnung durchzuführen, aber es ist auch eine unsichere Methode.