Document.addEventListener(‘DOMContentLoaded’, () => {

Const currentOperandTextElement = document.getElementById(‘current-operand’);

Const previousOperandTextElement = document.getElementById(‘previous-operand’);

Const themeToggleButton = document.getElementById(‘theme-toggle-btn’);

Const numberButtons = document.querySelectorAll(‘[data-number]’);

Const operationButtons = document.querySelectorAll(‘[data-operation]’);

Const scientificButtons = document.querySelectorAll(‘[data-sci-op]’);

Const modeToggleButtons = document.querySelectorAll(‘.mode-toggle’);

Const closePanelButtons = document.querySelectorAll(‘.close-panel-btn’);

Const panels = document.querySelectorAll(‘.panel’);

// Calculator state

Let currentOperand = ‘0’;

Let previousOperand = ‘’;

Let operation = undefined;

Let shouldResetScreen = false;

Let angleMode = ‘deg’; // ‘deg’ or ‘rad’

Function updateDisplay() {

currentOperandTextElement.innerText = formatDisplayNumber(currentOperand);

if (operation != null) {

previousOperandTextElement.innerText = `${formatDisplayNumber(previousOperand)} ${getOperationSymbol(operation)}`;

} else {

previousOperandTextElement.innerText = ‘’;

}

}

Function formatDisplayNumber(numberStr) {

If (numberStr === ‘’ || numberStr === null || numberStr === undefined) return ‘’;

If (numberStr === ‘Error’) return ‘Error’;

Const stringNumber = numberStr.toString();

Let integerDigits = parseFloat(stringNumber.split(‘.’)[0]);

Let decimalDigits = stringNumber.split(‘.’)[1];

Let integerDisplay;

If (isNaN(integerDigits)) {

integerDisplay = ‘’;

} else {

If (Math.abs(integerDigits) > 999999999999) { // 12 digits

Return parseFloat(numberStr).toExponential(5);

}

integerDisplay = integerDigits.toLocaleString(‘en’, { maximumFractionDigits: 0 });

}

If (decimalDigits != null) {

If ((integerDisplay + ‘.’ + decimalDigits).length > 16) {

Const availableDecimalPlaces = 16 – integerDisplay.length -1;

If (availableDecimalPlaces > 0) {

Return `${integerDisplay}.${decimalDigits.substring(0, availableDecimalPlaces)}`;

} else {

Return parseFloat(numberStr).toExponential(5);

}

}

Return `${integerDisplay}.${decimalDigits}`;

} else {

If (integerDisplay.length > 16) {

Return parseFloat(numberStr).toExponential(5);

}

Return integerDisplay;

}

}

Function getOperationSymbol(op) {

Const symbols = {

‘/’: ‘÷’,

‘\*’: ‘×’,

‘+’: ‘+’,

‘-‘: ‘-‘,

‘pow’: ‘^’

};

Return symbols[op] || ‘’;

}

Function appendNumber(number) {

If (currentOperand.includes(‘.’) && number === ‘.’) return;

If (shouldResetScreen || currentOperand === ‘0’ && number !== ‘.’) {

currentOperand = ‘’;

shouldResetScreen = false;

}

If (currentOperand.length > 15 && !shouldResetScreen) return;

currentOperand = currentOperand.toString() + number.toString();

}

Function chooseOperation(op) {

If (currentOperand === ‘’ && op !== ‘sqrt’ && op !== ‘inv’ && op !== ‘sqr’ && op !==’fact’ && op !==’abs’) return;

If (previousOperand !== ‘’) {

Compute();

}

Operation = op;

previousOperand = currentOperand;

currentOperand = ‘’;

shouldResetScreen = false;

}

Function compute() {

Let computation;

Const prev = parseFloat(previousOperand);

Const current = parseFloat(currentOperand);

If (isNaN(prev) || (isNaN(current) && operation !== ‘sqrt’ && operation !== ‘inv’ && operation !== ‘sqr’ && operation !==’fact’ && operation !==’abs’)) return;

Switch (operation) {

Case ‘+’: computation = prev + current; break;

Case ‘-‘: computation = prev – current; break;

Case ‘\*’: computation = prev \* current; break;

Case ‘/’:

If (current === 0) {

currentOperand = ‘Error’;

operation = undefined;

previousOperand = ‘’;

return;

}

Computation = prev / current;

Break;

Case ‘pow’: computation = Math.pow(prev, current); break;

Default: return;

}

currentOperand = isNaN(computation) ? ‘Error’ : computation.toString();

operation = undefined;

previousOperand = ‘’;

shouldResetScreen = true;

}

Function applyScientificOperation(sciOp) {

Let val = parseFloat(currentOperand);

If (isNaN(val) && sciOp !== ‘pi’ && sciOp !== ‘e’ && sciOp !== ‘(‘ && sciOp !== ‘)’) {

If (sciOp === ‘pi’) currentOperand = Math.PI.toString();

Else if (sciOp === ‘e’) currentOperand = Math.E.toString();

Else return;

} else {

Let result;

Const angle = angleMode === ‘deg’ ? (val \* Math.PI / 180) : val;

Switch (sciOp) {

Case ‘sin’: result = Math.sin(angle); break;

Case ‘cos’: result = Math.cos(angle); break;

Case ‘tan’: result = Math.tan(angle); break;

Case ‘asin’: result = Math.asin(val); if(angleMode === ‘deg’) result \*= 180/Math.PI; break;

Case ‘acos’: result = Math.acos(val); if(angleMode === ‘deg’) result \*= 180/Math.PI; break;

Case ‘atan’: result = Math.atan(val); if(angleMode === ‘deg’) result \*= 180/Math.PI; break;

Case ‘pi’: result = Math.PI; break;

Case ‘e’: result = Math.E; break;

Case ‘sqrt’: result = Math.sqrt(val); break;

Case ‘sqr’: result = Math.pow(val, 2); break;

Case ‘inv’: result = val === 0 ? ‘Error’ : 1 / val; break;

Case ‘log’: result = Math.log10(val); break;

Case ‘ln’: result = Math.log(val); break;

Case ‘fact’:

If (val < 0 || val !== Math.floor(val)) { result = ‘Error’; break; }

If (val === 0) { result = 1; break;}

If (val > 170) { result = ‘Infinity’; break; } // Factorial gets too large quickly

Let fact = 1;

For (let i = 1; i <= val; i++) fact \*= i;

Result = fact;

Break;

Case ‘abs’: result = Math.abs(val); break;

Case ‘deg’: angleMode = ‘deg’; updateAngleModeButtons(); return;

Case ‘rad’: angleMode = ‘rad’; updateAngleModeButtons(); return;

Case ‘(‘:

Case ‘)’:

appendNumber(sciOp);

updateDisplay();

return;

default: return;

}

currentOperand = isNaN(result) || !isFinite(result) ? ‘Error’ : result.toString();

}

shouldResetScreen = true;

updateDisplay();

}

Function updateAngleModeButtons() {

Const degButton = document.querySelector(‘[data-sci-op=”deg”]’);

Const radButton = document.querySelector(‘[data-sci-op=”rad”]’);

If (!degButton || !radButton) return;

Const isDarkMode = document.body.classList.contains(‘dark-mode’);

Const activeBg = isDarkMode ? ‘var(--special-button-bg-dark)’ : ‘var(--special-button-bg-light)’;

Const inactiveBg = isDarkMode ? ‘var(--button-bg-dark)’ : ‘var(--button-bg-light)’;

degButton.style.fontWeight = angleMode === ‘deg’ ? ‘bold’ : ‘normal’;

radButton.style.fontWeight = angleMode === ‘rad’ ? ‘bold’ : ‘normal’;

degButton.style.background = angleMode === ‘deg’ ? activeBg : inactiveBg;

radButton.style.background = angleMode === ‘rad’ ? activeBg : inactiveBg;

}

Function allClear() {

currentOperand = ‘0’;

previousOperand = ‘’;

operation = undefined;

shouldResetScreen = false;

}

Function clearEntry() {

currentOperand = ‘0’;

shouldResetScreen = false;

}

Function backspace() {

If (shouldResetScreen) {

allClear();

} else {

currentOperand = currentOperand.toString().slice(0, -1);

if (currentOperand === ‘’) currentOperand = ‘0’;

}

}

numberButtons.forEach(button => {

button.addEventListener(‘click’, () => {

appendNumber(button.dataset.number);

updateDisplay();

});

});

operationButtons.forEach(button => {

button.addEventListener(‘click’, () => {

const op = button.dataset.operation;

if (op === ‘equals’) {

compute();

} else if (op === ‘all-clear’) {

allClear();

} else if (op === ‘clear-entry’) {

clearEntry();

} else if (op === ‘backspace’) {

Backspace();

} else {

chooseOperation(op);

}

updateDisplay();

});

});

scientificButtons.forEach(button => {

button.addEventListener(‘click’, () => {

applyScientificOperation(button.dataset.sciOp);

});

});

themeToggleButton.addEventListener(‘click’, () => {

document.body.classList.toggle(‘dark-mode’);

if (document.body.classList.contains(‘dark-mode’)) {

themeToggleButton.textContent = ‘☀️’;

} else {

themeToggleButton.textContent = ‘🌙’;

}

updateAngleModeButtons();

});

modeToggleButtons.forEach(button => {

button.addEventListener(‘click’, () => {

const panelId = button.dataset.panelId;

const targetPanel = document.getElementById(panelId);

if (targetPanel) {

panels.forEach(p => { if (p.id !== panelId && p.classList.contains(‘active’)) p.classList.remove(‘active’);});

targetPanel.classList.toggle(‘active’);

}

});

});

closePanelButtons.forEach(button => {

button.addEventListener(‘click’, () => {

const panelId = button.dataset.panelId;

document.getElementById(panelId).classList.remove(‘active’);

});

});

// --- Unit Converter Logic ---

Const conversionTypeSelect = document.getElementById(‘conversion-type’);

Const converterGroups = {

Length: document.getElementById(‘length-converter’),

Weight: document.getElementById(‘weight-converter’),

Temperature: document.getElementById(‘temperature-converter’)

};

Const inputs = {

Meters: document.getElementById(‘meters’),

Centimeters: document.getElementById(‘centimeters’),

Kilograms: document.getElementById(‘kilograms’),

Grams: document.getElementById(‘grams’),

Celsius: document.getElementById(‘celsius’),

Kelvin: document.getElementById(‘kelvin’)

};

Function showConverterGroup(type) {

Object.values(converterGroups).forEach(group => { if(group) group.classList.add(‘hidden’)});

If (converterGroups[type]) {

converterGroups[type].classList.remove(‘hidden’);

}

}

If (conversionTypeSelect) {

conversionTypeSelect.addEventListener(‘change’, (e) => showConverterGroup(e.target.value));

showConverterGroup(‘length’); // Initial display

}

// Length

If (inputs.meters && inputs.centimeters) {

Inputs.meters.addEventListener(‘input’, (e) => {

Const m = parseFloat(e.target.value);

Inputs.centimeters.value = isNaN(m) ? ‘’ : (m \* 100).toFixed(2).replace(/\.00$/, ‘’);

});

Inputs.centimeters.addEventListener(‘input’, (e) => {

Const cm = parseFloat(e.target.value);

Inputs.meters.value = isNaN(cm) ? ‘’ : (cm / 100).toFixed(2).replace(/\.00$/, ‘’);

});

}

// Weight

If (inputs.kilograms && inputs.grams) {

Inputs.kilograms.addEventListener(‘input’, (e) => {

Const kg = parseFloat(e.target.value);

Inputs.grams.value = isNaN(kg) ? ‘’ : (kg \* 1000).toFixed(2).replace(/\.00$/, ‘’);

});

Inputs.grams.addEventListener(‘input’, (e) => {

Const g = parseFloat(e.target.value);

Inputs.kilograms.value = isNaN(g) ? ‘’ : (g / 1000).toFixed(3).replace(/\.000$/, ‘’);

});

}

// Temperature

If (inputs.celsius && inputs.kelvin) {

Inputs.celsius.addEventListener(‘input’, (e) => {

Const c = parseFloat(e.target.value);

Inputs.kelvin.value = isNaN(c) ? ‘’ : (c + 273.15).toFixed(2).replace(/\.00$/, ‘’);

});

Inputs.kelvin.addEventListener(‘input’, (e) => {

Const k = parseFloat(e.target.value);

Inputs.celsius.value = isNaN(k) ? ‘’ : (k – 273.15).toFixed(2).replace(/\.00$/, ‘’);

});

}

// --- Chemistry Tool: Molar Mass ---

Const chemicalFormulaInput = document.getElementById(‘chemical-formula’);

Const calcMolarMassBtn = document.getElementById(‘calculate-molar-mass-btn’);

Const molarMassResultEl = document.getElementById(‘molar-mass-result’);

Const atomicMasses = {

H: 1.008, O: 15.999, C: 12.011, N: 14.007, Na: 22.990, Cl: 35.453,

S: 32.06, P: 30.974, K: 39.098, Ca: 40.078, Fe: 55.845, Mg: 24.305,

// Add more common elements

};

If (calcMolarMassBtn) {

calcMolarMassBtn.addEventListener(‘click’, () => {

const formula = chemicalFormulaInput.value.trim();

if (!formula) {

molarMassResultEl.textContent = “Result: -- g/mol”;

return;

}

Try {

Const molarMass = calculateMolarMass(formula);

molarMassResultEl.textContent = `Result: ${molarMass.toFixed(3)} g/mol`;

} catch (error) {

molarMassResultEl.textContent = `Error: ${error.message}`;

}

});

}

Function calculateMolarMass(formula) {

Const elements = formula.match(/[A-Z][a-z]\*[0-9]\*/g);

If (!elements) throw new Error(“Invalid formula format”);

Let totalMass = 0;

Elements.forEach(elStr => {

Const match = elStr.match(/([A-Z][a-z]\*)(\d\*)/);

Const elementSymbol = match[1];

Const count = match[2] ? parseInt(match[2]) : 1;

If (!atomicMasses[elementSymbol]) {

Throw new Error(`Unknown element: ${elementSymbol}`);

}

totalMass += atomicMasses[elementSymbol] \* count;

});

Return totalMass;

}

// Initial setup

allClear();

updateDisplay();

updateAngleModeButtons();

});