```
/**
 * demo.js
 * Small documented JavaScript module for the "Demonstration & Documentation" project.
 *
 * Usage:
 *  - Open index.html in a browser (module import)
 *  - Buttons demonstrate functions and async behavior
 *
 * Author: Your Name
 * Version: 1.0
 */

/**
 * Adds two numbers.
 * Note: performs numeric coercion; inputs should be convertible to numbers.
 * @param {number|string} a
 * @param {number|string} b
 * @returns {number} sum of a and b
 */
export function add(a, b) {
  const x = Number(a);
  const y = Number(b);
  if (Number.isNaN(x) || Number.isNaN(y)) {
    throw new TypeError('add: inputs must be numbers or numeric strings');
  }
  return x + y;
}

/**
 * Multiplies two numbers.
 * @param {number|string} a
```

```
 * @param {number|string} b
 * @returns {number}
 */
export function multiply(a, b) {
  const x = Number(a);
  const y = Number(b);
  if (Number.isNaN(x) || Number.isNaN(y)) {
    throw new TypeError('multiply: inputs must be numbers or numeric strings');
  }
  return x * y;
}

/**
 * Simulates a fetch call returning JSON after a delay.
 * Useful for demonstrating async/await error handling.
 * @param {Object} payload - object to "return"
 * @param {number} delayMs - delay in milliseconds
 * @param {boolean} [shouldFail=false] - whether to simulate a failure
 * @returns {Promise<Object>}
 */
export function simulateFetch(payload = {}, delayMs = 800, shouldFail = false) {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      if (shouldFail) {
        reject(new Error('Simulated network error'));
      } else {
        resolve({
          ok: true,
          timestamp: new Date().toISOString(),
          data: payload,
        });
```

```
    }
  }, Math.max(0, Number(delayMs) || 0));
 });
}

/**
 * Simple logger that writes to the page log area if available and console.
 * @param {string} message
 */
export function pageLog(message) {
 const el = document.getElementById('log');
 const time = new Date().toLocaleTimeString();
 const formatted = [${time}] ${message};
 if (el) {
  // Prepend so the newest message appears at top
  el.textContent = formatted + '\n' + el.textContent.replace(/^Console log output will appear here...\n?/, '');
 }
 console.log(formatted);
}

/* ---------------------------
  Wire up UI (index.html)
  --------------------------- */
if (typeof window !== 'undefined') {
 const $ = id => document.getElementById(id);
 const numA = $('numA');
 const numB = $('numB');
 const resultEl = $('result');

 $('addBtn').addEventListener('click', () => {
```

```
  try {
    const res = add(numA.value, numB.value);
    resultEl.textContent = String(res);
    pageLog(add(${numA.value}, ${numB.value}) -> ${res});
  } catch (err) {
    resultEl.textContent = 'Error';
    pageLog(ERROR: ${err.message});
  }
});

$('multiplyBtn').addEventListener('click', () => {
  try {
    const res = multiply(numA.value, numB.value);
    resultEl.textContent = String(res);
    pageLog(multiply(${numA.value}, ${numB.value}) -> ${res});
  } catch (err) {
    resultEl.textContent = 'Error';
    pageLog(ERROR: ${err.message});
  }
});

$('simulateBtn').addEventListener('click', async () => {
  resultEl.textContent = 'loading...';
  pageLog('Starting simulated fetch...');
  try {
    const resp = await simulateFetch({ a: numA.value, b: numB.value }, 1000, false);
    resultEl.textContent = JSON.stringify(resp.data);
    pageLog(simulateFetch success @ ${resp.timestamp});
  } catch (err) {
    resultEl.textContent = 'Fetch failed';
    pageLog(simulateFetch ERROR: ${err.message});
```

```
    }
  });

  $('clearBtn').addEventListener('click', () => {
    const l = $('log');
    if (l) l.textContent = 'Console log output will appear here...';
    pageLog('Log cleared');
  });
}
```