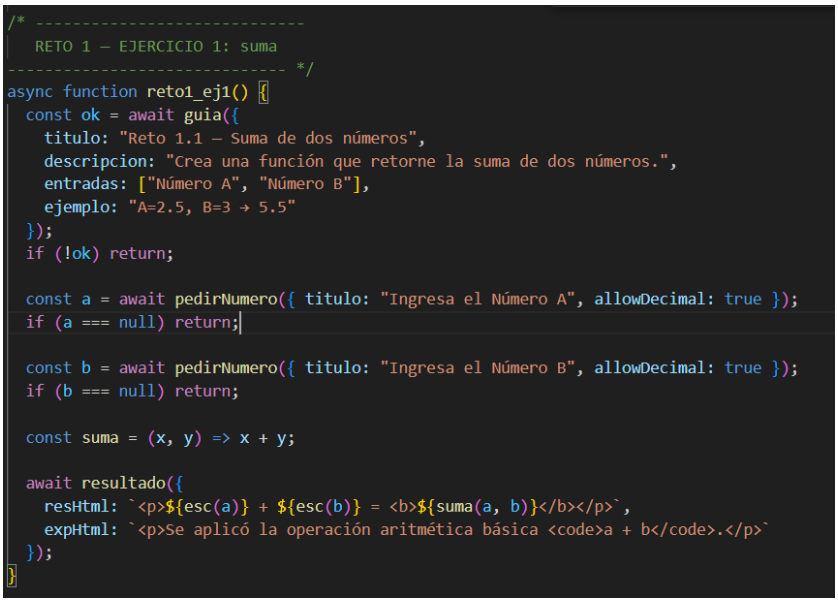
SV73873639

Reto 1 — (5 Ejercicios)

Reto 1- Ejercicio 1: Suma de dos números

Código:



```
/* ----------------------------
   RETO 1 — EJERCICIO 1: suma
---------------------------- */
async function reto1_ej1() {
  const ok = await guia({
    titulo: "Reto 1.1 — Suma de dos números",
    descripcion: "Crea una función que retorne la suma de dos números.",
    entradas: ["Número A", "Número B"],
    ejemplo: "A=2.5, B=3 → 5.5"
  });
  if (!ok) return;

  const a = await pedirNumero({ titulo: "Ingresa el Número A", allowDecimal: true });
  if (a === null) return;

  const b = await pedirNumero({ titulo: "Ingresa el Número B", allowDecimal: true });
  if (b === null) return;

  const suma = (x, y) => x + y;

  await resultado({
    resHtml: `<p>${esc(a)} + ${esc(b)} = <b>${suma(a, b)}</b></p>`,
    expHtml: `<p>Se aplicó la operación aritmética básica <code>a + b</code>.</p>`
  });
}
```

Explicación:

Este ejercicio utiliza una **arrow function** (función de flecha) para realizar la operación aritmética. La captura de datos es **asíncrona**, asegurando que las variables a y b contengan valores válidos antes de ejecutar la suma. Se emplea **template literals** (plantillas de cadena) para construir el resultado HTML final.

Procedimiento:

1. Se muestra la guía del ejercicio con **guia()** y se valida confirmación.
2. Se solicita el Número A mediante **pedirNumero()** con decimales permitidos.
3. Se solicita el Número B mediante **pedirNumero()** con decimales permitidos.
4. Se define la función suma usando sintaxis de flecha.
5. Se calcula **suma(a, b)** con los valores ingresados.
6. Se muestra el resultado usando **resultado()**.

Reto 1 — Ejercicio 2: Potencia de un número

Código:

```
/* ----------------------------
   RETO 1 — EJERCICIO 2: potencia
---------------------------- */
async function reto1_ej2() {
  const ok = await guia({
    titulo: "Reto 1.2 — Potencia de un número",
    descripcion: "Crea una función que retorne la potencia de un número dado. Recibe número y potencia.",
    entradas: ["Base (número)", "Exponente (número)"],
    ejemplo: "base=2, exponente=3 → 8"
  });
  if (!ok) return;

  const base = await pedirNumero({ titulo: "Ingresa la base", allowDecimal: true });
  if (base === null) return;

  const exp = await pedirNumero({ titulo: "Ingresa el exponente", allowDecimal: true });
  if (exp === null) return;

  const potencia = (b, e) => Math.pow(b, e);

  await resultado({
    resHtml: `<p>${esc(base)} ^ ${esc(exp)} = <b>${potencia(base, exp)}</b></p>`,
    expHtml: `<p>Se utilizó <code>Math.pow(base, exponente)</code> para calcular la potencia.</p>`
  });
}
```

Explicación:

Este ejercicio calcula una potencia usando Math.pow(base, exponente), encapsulado dentro de una función flecha para separar la lógica del flujo interactivo. async/await mantiene la captura de base y exponente en orden, y el resultado se presenta con interpolación en el modal..

Procedimiento:

1. Se presenta la guía con guia() y se valida confirmación.
2. Se solicita la base con pedirNumero().
3. Se solicita el exponente con pedirNumero().
4. Se define la función potencia usando Math.pow().
5. Se calcula potencia(base, exp).
6. Se muestra el resultado con resultado().

Reto 1 — Ejercicio 3: Suma de cubos

Código:

```
/* ------------------------------
   RETO 1 — EJERCICIO 3: suma de cubos
------------------------------ */
async function reto1_ej3() {
  const ok = await guia({
    titulo: "Reto 1.3 — Suma de cubos",
    descripcion: "Crea una función que tome números y devuelva la suma de sus cubos.",
    entradas: ["Número 1", "Número 2", "Número 3"],
    ejemplo: "sumOfCubes(1, 5, 9) → 855"
  });
  if (!ok) return;

  const n1 = await pedirNumero({ titulo: "Número 1", allowDecimal: true });
  if (n1 === null) return;

  const n2 = await pedirNumero({ titulo: "Número 2", allowDecimal: true });
  if (n2 === null) return;

  const n3 = await pedirNumero({ titulo: "Número 3", allowDecimal: true });
  if (n3 === null) return;

  const sumOfCubes = (...nums) => nums.reduce((acc, n) => acc + n ** 3, 0);
  const r = sumOfCubes(n1, n2, n3);

  await resultado({
    resHtml: `<p>${esc(n1)}³ + ${esc(n2)}³ + ${esc(n3)}³ = <b>${r}</b></p>`,
    expHtml: `<p>Se elevó cada número al cubo (<code>n³</code>) y luego se sumaron los resultados.</p>`
  });
}
```

- **Explicación:**
  Este ejercicio suma los cubos de tres números usando una función con Rest Parameters (...nums) y acumulación con reduce(). Cada número se eleva al cubo con el operador **, y el total se calcula de forma funcional y directa antes de mostrarse al usuario..
- **Procedimiento:**
  1. Se muestra la guía y se valida confirmación.
  2. Se solicita el Número 1 con pedirNumero().
  3. Se solicita el Número 2 con pedirNumero().
  4. Se solicita el Número 3 con pedirNumero().
  5. Se define sumOfCubes(...nums) usando REST y reduce().
  6. Se calcula r = sumOfCubes(n1, n2, n3).
  7. Se presenta el resultado con resultado().

Reto 1 — Ejercicio 4: Área de triángulo

Código:

```
/* ----------------------------
   RETO 1 — EJERCICIO 4: triArea
---------------------------- */
async function reto1_ej4() {
  const ok = await guia({
    titulo: "Reto 1.4 — Área de triángulo",
    descripcion: "Toma base y altura de un triángulo y devuelve su área.",
    entradas: ["Base (≥ 0)", "Altura (≥ 0)"],
    ejemplo: "triArea(3, 2) → 3"
  });
  if (!ok) return;

  const b = await pedirNumero({ titulo: "Base", allowDecimal: true, allowNegative: false });
  if (b === null) return;

  const h = await pedirNumero({ titulo: "Altura", allowDecimal: true, allowNegative: false });
  if (h === null) return;

  const triArea = (base, altura) => (base * altura) / 2;

  await resultado({
    resHtml: `<p>Área = ( ${esc(b)} × ${esc(h)} ) / 2 = <b>${triArea(b, h)}</b></p>`,
    expHtml: `<p>Se aplicó la fórmula geométrica estándar: <code>(base×altura)/2</code>.</p>`
  });
}
```

Explicación:

Este ejercicio calcula el área de un triángulo aplicando (base * altura) / 2 dentro de una arrow function. La validación impide números negativos usando allowNegative: false, lo cual asegura que los datos estén en un dominio válido antes de ejecutar el cálculo.

Procedimiento:

1. Se presenta la guía y se valida confirmación.
2. Se solicita la base con **pedirNumero()** evitando negativos.
3. Se solicita la altura con **pedirNumero()** evitando negativos.
4. Se define **triArea** = (base, altura) => (base * altura) / 2.
5. Se calcula **triArea(b, h)**.
6. Se muestra el resultado con **resultado()**.

Reto 1 — Ejercicio 5: calculator (Switch)

- ## Código:

```
/* ----------------------------
   RETO 1 — EJERCICIO 5: calculator (switch)
---------------------------- */
async function reto1_ej5() {
  const ok = await guia({
    titulo: "Reto 1.5 — calculator (Switch)",
    descripcion: "Recibe dos números y una operación matemática (+, -, *, /, %).",
    entradas: ["Número A", "Operación", "Número B"],
    ejemplo: 'calculator(2, "+", 2) → 4'
  });
  if (!ok) return;

  const a = await pedirNumero({ titulo: "Número A", allowDecimal: true });
  if (a === null) return;

  const opRes = await Swal.fire({
    title: "Elige la operación",
    input: "select",
    inputOptions: {
      "+": "Suma (+)",
      "-": "Resta (-)",
      "*": "Multiplicación (*)",
      "/": "División (/)",
      "%": "Módulo (%)"
    },
    showCancelButton: true,
    confirmButtonText: "Aceptar",
    cancelButtonText: "Cancelar",
    confirmButtonColor: "#4f46e5",
    didOpen: enableEnterConfirm,
    inputValidator: (v) => (v ? undefined : "Selecciona una operación.")
  });
  if (!opRes.isConfirmed) return;

  const op = opRes.value;

  const b = await pedirNumero({ titulo: "Número B", allowDecimal: true });
  if (b === null) return;

  const calculator = (x, oper, y) => {
    switch (oper) {
      case "+": return x + y;
      case "-": return x - y;
      case "*": return x * y;
      case "/": return y !== 0 ? x / y : "División por 0 inválida";
      case "%": return y !== 0 ? x % y : "Módulo por 0 inválido";
      default: return "El parámetro no es reconocido";
    }
  };

  const r = calculator(a, op, b);

  await resultado({
    resHtml: `<p>${esc(a)} ${esc(op)} ${esc(b)} = <b>${esc(r)}</b></p>`,
    expHtml: `<p>Se usó <code>switch</code> para seleccionar la operación y retornar el cálculo.</p>`
  });
}
```

- ## Explicación:

  Este ejercicio realiza operaciones básicas seleccionadas por el usuario usando switch. Para división (/) y módulo (%) se evita el error de dividir entre cero con un operador ternario que retorna un mensaje si y === 0. La operación se elige desde un select en SweetAlert2 y luego se procesa según el caso..

- ## Procedimiento:

  1. Se muestra la guía del ejercicio y se valida confirmación.
  2. Se solicita el Número A con pedirNumero().
  3. Se muestra un selector de operación con Swal.fire({ input: "select" }).
  4. Se valida que el usuario haya elegido una operación.
  5. Se solicita el Número B con pedirNumero().
  6. Se define calculator(x, oper, y) con switch.
  7. Se calcula r = calculator(a, op, b).
  8. Se muestra el resultado con resultado().

Reto 2 — (22 Ejercicios)

Reto 2 — Ejercicio 1: Saludo (Arrow + Template)

Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 1: saludo
---------------------------- */
async function reto2_ej1() {
  const ok = await guia({
    titulo: "Reto 2.1 — Saludo (Arrow + Template)",
    descripcion: "Recibe nombre, apellido y edad y retorna un string concatenado.",
    entradas: ["Nombre (solo letras)", "Apellido (solo letras)", "Edad (entero)"],
    ejemplo: "Hola mi nombre es Sebastián Yabiku y mi edad 33"
  });
  if (!ok) return;

  const nombre = await pedirTexto({
    titulo: "Nombre",
    placeholder: "Ej.: Sebastián",
    validate: (v) => (/^[a-zA-ZáéíóúÁÉÍÓÚñÑ\s]+$/.test(v) ? undefined : "Solo letras y espacios.")
  });
  if (nombre === null) return;

  const apellido = await pedirTexto({
    titulo: "Apellido",
    placeholder: "Ej.: Yabiku",
    validate: (v) => (/^[a-zA-ZáéíóúÁÉÍÓÚñÑ\s]+$/.test(v) ? undefined : "Solo letras y espacios.")
  });
  if (apellido === null) return;

  const edad = await pedirNumero({ titulo: "Edad (entero)", allowDecimal: false, allowNegative: false, placeholder: "Ej.: 33" });
  if (edad === null) return;

  const saludo = (n, a, e) => `Hola mi nombre es ${n} ${a} y mi edad ${e}`;
  const frase = saludo(nombre, apellido, edad);

  await resultado({
    resHtml: `<p><b>${esc(frase)}</b></p>`,
    expHtml: `<p>Se construyó el texto con <i>template literals</i>: <code>\`...\${variable}...\`</code>.</p>`
  });
}
```

Explicación:

Este ejercicio construye una cadena dinámica utilizando una **arrow function** y **template literals** para interpolar nombre, apellido y edad dentro del texto. Se emplea async/await para manejar la entrada de datos de manera secuencial y validaciones con expresiones regulares para asegurar que nombre y apellido contengan solo letras. El resultado se genera de forma clara y segura antes de mostrarse en el modal.

Procedimiento:

1. Se muestra la guía informativa del ejercicio.
2. Se solicita el nombre con validación alfabética.
3. Se solicita el apellido con la misma validación.
4. Se solicita la edad como número entero no negativo.
5. Se define la función flecha saludo.
6. Se ejecuta la función con los datos ingresados.
7. Se presenta el mensaje final en el modal.

Reto 2 — Ejercicio 2: sumOfCubes

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 2: sumOfCubes
---------------------------- */
async function reto2_ej2() {
  const ok = await guia({
    titulo: "Reto 2.2 — sumOfCubes",
    descripcion: "Toma números y devuelve la suma de sus cubos.",
    entradas: ["Lista de números (coma o espacio)"],
    ejemplo: "sumOfCubes(1, 5, 9) → 855"
  });
  if (!ok) return;

  const nums = await pedirListaNumeros({ titulo: "Ingresa los números", placeholder: "Ej.: 1, 5, 9" });
  if (!nums) return;

  const sumOfCubes = (...args) => args.reduce((acc, n) => acc + n ** 3, 0);
  const r = sumOfCubes(...nums);

  await resultado({
    resHtml: `<p>Números: <code>[${esc(nums.join(", "))}]</code></p><p>Σ(n³) = <b>${r}</b></p>`,
    expHtml: `<p>Se elevó cada número al cubo y se sumaron los cubos con <code>reduce</code>.</p>`
  });
}
```

- Explicación:

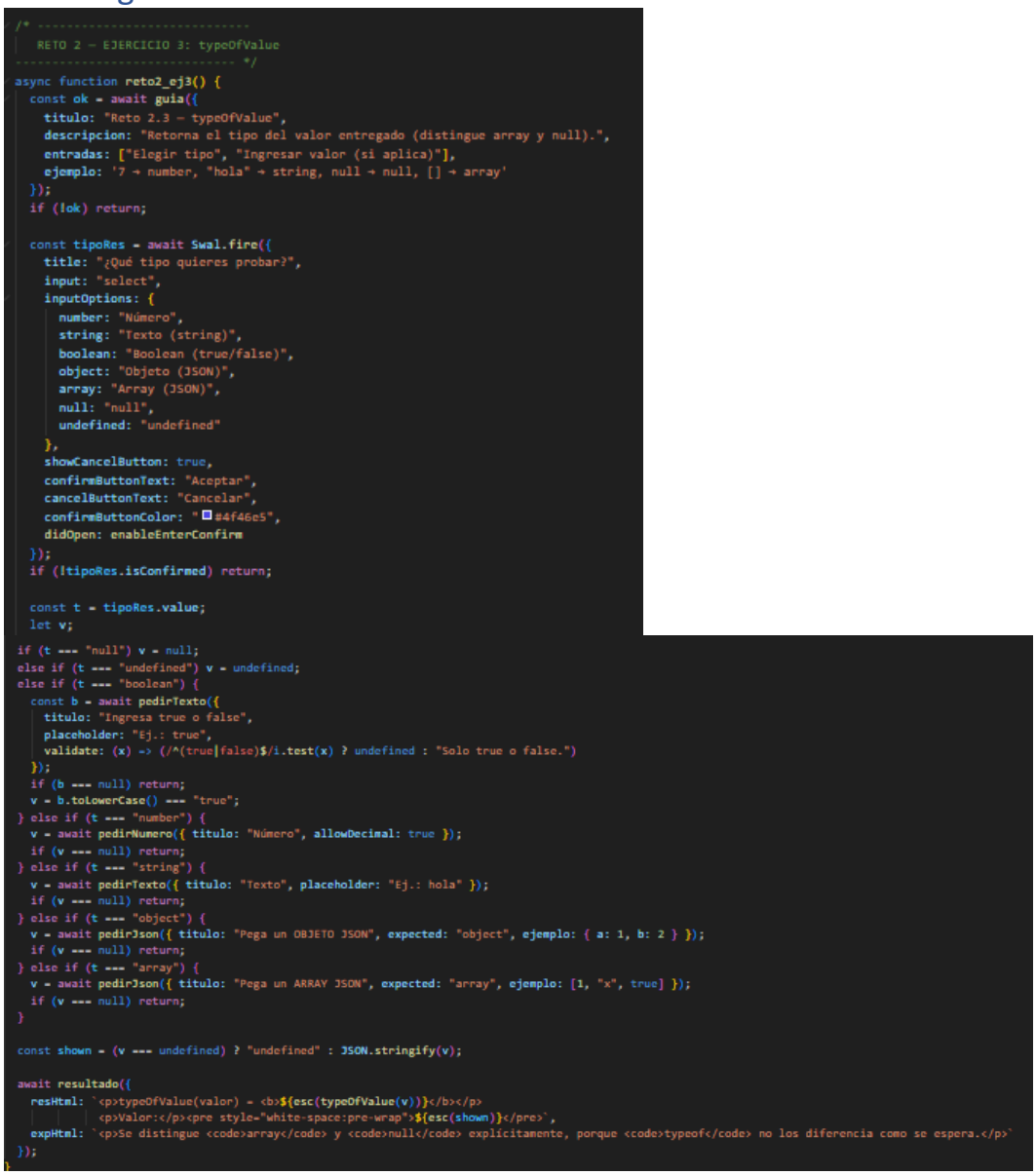
Este ejercicio utiliza **Rest Parameters** (...args) para recibir múltiples números como un arreglo y el método reduce() para calcular la suma de sus cubos. El operador de exponenciación (**) eleva cada número a la potencia 3. El uso de async/await permite capturar los datos antes de procesarlos, manteniendo el flujo ordenado y validado.

• Procedimiento:

1. Se muestra la guía del ejercicio.
2. Se solicita una lista de números separados por coma o espacio.
3. Se valida y convierte la entrada en un arreglo numérico.
4. Se define la función sumOfCubes usando ...args.
5. Se aplica reduce() para sumar n ** 3 desde un acumulador inicial de 0.
6. Se guarda el resultado en una variable.
7. Se muestra el resultado final en el modal.

Reto 2 — Ejercicio 3: typeOfValue

- Código:



```
/* ----------------------------
   RETO 2 — EJERCICIO 3: typeOfValue
---------------------------- */
async function reto2_ej3() {
  const ok = await guia({
    titulo: "Reto 2.3 — typeOfValue",
    descripcion: "Retorna el tipo del valor entregado (distingue array y null).",
    entradas: ["Elegir tipo", "Ingresar valor (si aplica)"],
    ejemplo: '7 → number, "hola" → string, null → null, [] → array'
  });
  if (!ok) return;

  const tipoRes = await Swal.fire({
    title: "¿Qué tipo quieres probar?",
    input: "select",
    inputOptions: {
      number: "Número",
      string: "Texto (string)",
      boolean: "Boolean (true/false)",
      object: "Objeto (JSON)",
      array: "Array (JSON)",
      null: "null",
      undefined: "undefined"
    },
    showCancelButton: true,
    confirmButtonText: "Aceptar",
    cancelButtonText: "Cancelar",
    confirmButtonColor: "#4f46e5",
    didOpen: enableEnterConfirm
  });
  if (!tipoRes.isConfirmed) return;

  const t = tipoRes.value;
  let v;

if (t === "null") v = null;
else if (t === "undefined") v = undefined;
else if (t === "boolean") {
  const b = await pedirTexto({
    titulo: "Ingresa true o false",
    placeholder: "Ej.: true",
    validate: (x) => (/^(true|false)$/i.test(x) ? undefined : "Solo true o false.")
  });
  if (b === null) return;
  v = b.toLowerCase() === "true";
} else if (t === "number") {
  v = await pedirNumero({ titulo: "Número", allowDecimal: true });
  if (v === null) return;
} else if (t === "string") {
  v = await pedirTexto({ titulo: "Texto", placeholder: "Ej.: hola" });
  if (v === null) return;
} else if (t === "object") {
  v = await pedirJson({ titulo: "Pega un OBJETO JSON", expected: "object", ejemplo: { a: 1, b: 2 } });
  if (v === null) return;
} else if (t === "array") {
  v = await pedirJson({ titulo: "Pega un ARRAY JSON", expected: "array", ejemplo: [1, "x", true] });
  if (v === null) return;
}

const shown = (v === undefined) ? "undefined" : JSON.stringify(v);

await resultado({
  resHtml: `<p>typeOfValue(valor) = <b>${esc(typeOfValue(v))}</b></p>
            <p>Valor:</p><pre style="white-space:pre-wrap">${esc(shown)}</pre>`,
  expHtml: `<p>Se distingue <code>array</code> y <code>null</code> explícitamente, porque <code>typeof</code> no los diferencia como se espera.</p>`
});
}
```

- Explicación:

  La función mejora el comportamiento del operador typeof, ya que este identifica tanto null como los arreglos como "object". Para evitar esa ambigüedad, se utiliza Array.isArray() para detectar arreglos y una comparación estricta (===) para identificar null, garantizando una clasificación más precisa del tipo de dato.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. El usuario selecciona el tipo de dato que desea evaluar.
  3. Se solicita el valor correspondiente según el tipo elegido.
  4. Se verifica si el valor es null.
  5. Se verifica si es un arreglo mediante Array.isArray().
  6. En otros casos, se aplica typeof.
  7. Se muestra el tipo detectado junto con el valor ingresado.

Reto 2 — Ejercicio 4: Suma con REST

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 4: REST suma
---------------------------- */
async function reto2_ej4() {
  const ok = await guia({
    titulo: "Reto 2.4 — Suma con REST",
    descripcion: "Recibe n cantidad de argumentos y los suma.",
    entradas: ["Lista de números (coma o espacio)"],
    ejemplo: "10, 20, 5 → 35"
  });
  if (!ok) return;

  const nums = await pedirListaNumeros({ titulo: "Ingresa los números", placeholder: "Ej.: 10, 20, 5" });
  if (!nums) return;

  const sumarTodo = (...args) => args.reduce((acc, curr) => acc + Number(curr), 0);
  const r = sumarTodo(...nums);

  await resultado({
    resHtml: `<p>Suma = <b>${r}</b></p>`,
    expHtml: `<p><code>...args</code> captura una cantidad variable y <code>reduce</code> acumula la suma.</p>`
  });
}
```

- Explicación:

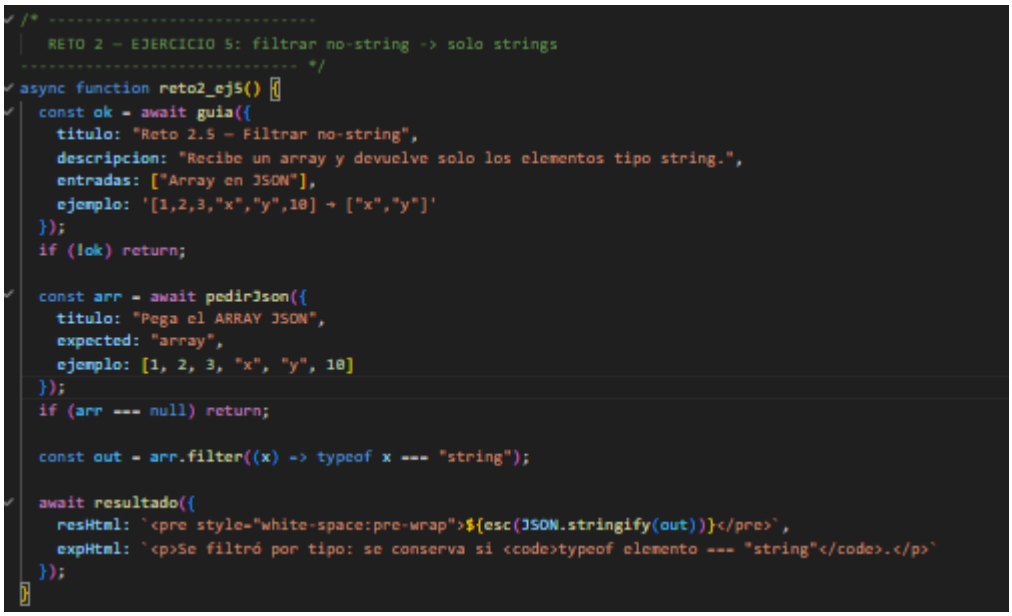
Este ejercicio suma una cantidad variable de números usando **Rest Parameters** (...args) para capturarlos en un arreglo y reduce() para acumular la suma. Además, se aplica Number(curr) para asegurar conversión numérica durante la acumulación, evitando concatenaciones accidentales si algún valor llegara como string.

- Procedimiento:

1. Se muestra la guía del ejercicio.
2. Se solicita una lista de números separados por coma o espacio.
3. Se valida y transforma la entrada en un arreglo numérico.
4. Se define sumarTodo(...args) para capturar todos los valores.
5. Se ejecuta reduce() para sumar, iniciando el acumulador en 0.
6. Se obtiene el total y se guarda en r.
7. Se presenta el resultado en el modal.

Reto 2 — Ejercicio 5: Filtrar no-string → solo strings

- Código:



```
/* ----------------------------
   RETO 2 — EJERCICIO 5: filtrar no-string -> solo strings
   ---------------------------- */
async function reto2_ej5() {
  const ok = await guia({
    titulo: "Reto 2.5 — Filtrar no-string",
    descripcion: "Recibe un array y devuelve solo los elementos tipo string.",
    entradas: ["Array en JSON"],
    ejemplo: '[1,2,3,"x","y",10] → ["x","y"]'
  });
  if (!ok) return;

  const arr = await pedirJson({
    titulo: "Pega el ARRAY JSON",
    expected: "array",
    ejemplo: [1, 2, 3, "x", "y", 10]
  });
  if (arr === null) return;

  const out = arr.filter((x) => typeof x === "string");

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Se filtró por tipo: se conserva si <code>typeof elemento === "string"</code>.</p>`
  });
}
```

- Explicación:

  Este ejercicio filtra un arreglo dejando únicamente elementos de tipo string mediante filter() y typeof. Es un ejemplo directo de procesamiento funcional: en vez de modificar el arreglo original, se construye uno nuevo con los elementos que cumplen la condición.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita un arreglo en formato JSON.
  3. Se valida que el dato ingresado sea un array.
  4. Se aplica filter(x => typeof x === "string").
  5. Se genera el arreglo de salida con solo strings.
  6. Se muestra el resultado en el modal.

Reto 2 — Ejercicio 6: minMax

- ## Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 6: minMax
---------------------------- */
async function reto2_ej6() {
  const ok = await guia({
    titulo: "Reto 2.6 — minMax",
    descripcion: "Toma una matriz de números y devuelve [mínimo, máximo].",
    entradas: ["Lista numérica (coma o espacio)"],
    ejemplo: "minMax([1,2,3,4,5]) → [1,5]"
  });
  if (!ok) return;

  const nums = await pedirListaNumeros({ titulo: "Ingresa los números", placeholder: "Ej.: 1,2,3,4,5" });
  if (!nums) return;

  const res = [Math.min(...nums), Math.max(...nums)];

  await resultado({
    resHtml: `<p>minMax = <b>[${esc(res.join(", "))}]</b></p>`,
    expHtml: `<p>Se usó <code>Math.min</code> y <code>Math.max</code> con spread (<code>...</code>).</p>`
  });
}
```

- ## Explicación:

Este ejercicio obtiene el mínimo y el máximo de una lista usando Math.min y Math.max. Se emplea el **spread operator** (...nums) para expandir el arreglo como argumentos, ya que estas funciones no reciben arrays directamente.

- ## Procedimiento:

1. Se muestra la guía del ejercicio.
2. Se solicita una lista de números.
3. Se valida y convierte a arreglo numérico.
4. Se calcula el mínimo con Math.min(...nums).
5. Se calcula el máximo con Math.max(...nums).
6. Se construye el resultado [min, max].
7. Se muestra el arreglo final en el modal.

Reto 2 — Ejercicio 7: formatPhoneNumber

- ## Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 7: formatPhoneNumber
---------------------------- */
async function reto2_ej7() {
  const ok = await guia({
    titulo: "Reto 2.7 — formatPhoneNumber",
    descripcion: "Toma 10 enteros (0–9) y devuelve un string con formato de teléfono.",
    entradas: ["10 dígitos (1234567890 o 1,2,3,...)"],
    ejemplo: "(123) 456-7890"
  });
  if (!ok) return;

  const raw = await pedirTexto({
    titulo: "Ingresa 10 dígitos",
    placeholder: "Ej.: 1234567890  (o 1,2,3,4,5,6,7,8,9,0)"
  });
  if (raw === null) return;

  let digits;
  if (/^\d{10}$/.test(raw)) digits = raw.split("").map(Number);
  else digits = raw.split(/[\s,]+/).filter(Boolean).map(Number);

  const okDigits = Array.isArray(digits)
    && digits.length === 10
    && digits.every((d) => Number.isInteger(d) && d >= 0 && d <= 9);

  if (!okDigits) {
    await Swal.fire({ icon: "error", title: "Dato inválido", text: "Debes ingresar exactamente 10 dígitos (0 a 9)." });
    return;
  }

  const tel = formatPhoneNumber(digits);

  await resultado({
    resHtml: `<p><b>${esc(tel)}</b></p>`,
    expHtml: `<p>Se agruparon los dígitos en (3)-(3)-(4) y se concatenó el formato.</p>`
  });
}
```

- ## Explicación:

  Este ejercicio valida que existan exactamente 10 dígitos (0–9) y luego los formatea como teléfono. Usa slice() para segmentar el arreglo en grupos (3-3-4) y join("") para convertir cada grupo a texto, construyendo el formato final con template literals.

- ## Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicitan 10 dígitos (como cadena continua o separados).
  3. Se convierte la entrada a un arreglo de números.
  4. Se valida que sean 10 elementos y que cada uno esté entre 0 y 9.
  5. Se formatea con formatPhoneNumber(digits) usando slice() y join().
  6. Se obtiene el string con formato telefónico.
  7. Se muestra el resultado en el modal.

Reto 2 — Ejercicio 8: findLargestNums

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 8: findLargestNums
---------------------------- */
async function reto2_ej8() {
  const ok = await guia({
    titulo: "Reto 2.8 — findLargestNums",
    descripcion: "Toma una matriz de matrices numéricas y retorna el mayor de cada sub-array.",
    entradas: ["Matriz en JSON (array de arrays)"],
    ejemplo: "[[4,2,7,1],[20,70,40,90],[1,2,0]] → [7,90,2]"
  });
  if (!ok) return;

  const mat = await pedirJson({
    titulo: "Pega la MATRIZ en JSON",
    expected: "array",
    ejemplo: [[4, 2, 7, 1], [20, 70, 40, 90], [1, 2, 0]]
  });
  if (mat === null) return;

  const valid = Array.isArray(mat)
    && mat.length > 0
    && mat.every((row) => Array.isArray(row) && row.length > 0 && row.every((x) => !Number.isNaN(Number(x))));

  if (!valid) {
    await Swal.fire({ icon: "error", title: "Dato inválido", text: "Debe ser un array de arrays numéricos." });
    return;
  }

  const out = mat.map((row) => Math.max(...row.map(Number)));

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Se obtuvo el máximo de cada sub-array con <code>Math.max</code>.</p>`
  });
}
```

- Explicación:

Este ejercicio recibe una matriz (array de arrays) y retorna el mayor valor de cada subarreglo. Se usa map() para transformar cada fila y Math.max(...row) para obtener el máximo, combinando programación funcional con spread operator.

- Procedimiento:

1. Se muestra la guía del ejercicio.
2. Se solicita una matriz en formato JSON (array de arrays).
3. Se valida que cada fila sea un array y contenga valores numéricos.
4. Se aplica map() para procesar cada subarreglo.
5. En cada fila, se calcula el máximo con Math.max(...row.map(Number)).
6. Se construye el arreglo final con máximos por fila.
7. Se muestra el resultado en el modal.

Reto 2 — Ejercicio 9: charIndex

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 9: charIndex
---------------------------- */
async function reto2_ej9() {
  const ok = await guia({
    titulo: "Reto 2.9 — charIndex",
    descripcion: "Devuelve el primer y último índice de un carácter dentro de una palabra.",
    entradas: ["Palabra", "Carácter (1)"],
    ejemplo: 'charIndex("hello","l") → [2,3]'
  });
  if (!ok) return;

  const word = await pedirTexto({ titulo: "Palabra", placeholder: "Ej.: hello" });
  if (word === null) return;

  const ch = await pedirTexto({
    titulo: "Carácter (solo 1)",
    placeholder: "Ej.: l",
    validate: (v) => (v.length === 1 ? undefined : "Debe ser un solo carácter.")
  });
  if (ch === null) return;

  const first = word.indexOf(ch);
  const last = word.lastIndexOf(ch);

  await resultado({
    resHtml: first === -1 ? `<p>El carácter no aparece.</p>` : `<p>Salida: <b>[${first}, ${last}]</b></p>`,
    expHtml: `<p>Se usó <code>indexOf</code> y <code>lastIndexOf</code>.</p>`
  });
}
```

- Explicación:

Este ejercicio obtiene el primer y último índice de un carácter dentro de una palabra usando indexOf() y lastIndexOf(). Si el carácter no existe, indexOf() retorna -1, lo que se usa como condición para informar que no aparece.

- Procedimiento:

1. Se muestra la guía del ejercicio.
2. Se solicita una palabra al usuario.
3. Se solicita un carácter y se valida que tenga longitud 1.
4. Se calcula first = word.indexOf(ch).
5. Se calcula last = word.lastIndexOf(ch).
6. Se evalúa si first === -1 para detectar ausencia.
7. Se muestra [first, last] o el mensaje correspondiente.

Reto 2 — Ejercicio 10: toArray

- Código:

```
/* ------------------------------
   RETO 2 — EJERCICIO 10: toArray
------------------------------ */
async function reto2_ej10() {
  const ok = await guia({
    titulo: "Reto 2.10 — toArray",
    descripcion: "Convierte un objeto en una matriz [clave, valor].",
    entradas: ["Objeto en JSON"],
    ejemplo: '{"a":1,"b":2} → [["a",1],["b",2]]'
  });
  if (!ok) return;

  const obj = await pedirJson({ titulo: "Pega el OBJETO JSON", expected: "object", ejemplo: { a: 1, b: 2 } });
  if (obj === null) return;

  const out = Object.entries(obj);

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Se aplicó <code>Object.entries</code> para obtener pares clave-valor.</p>`
  });
}
```

- Explicación:

  Este ejercicio convierte un objeto en un arreglo de pares [clave, valor] usando Object.entries(). Es una forma estándar de transformar objetos para poder iterarlos como estructura tabular.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita un objeto en formato JSON.
  3. Se valida que el JSON corresponda a un objeto (no array ni null).
  4. Se ejecuta Object.entries(obj).
  5. Se obtiene el arreglo de pares clave-valor.
  6. Se muestra el resultado en el modal.

Reto 2 — Ejercicio 11: getBudgets

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 11: getBudgets
---------------------------- */
async function reto2_ej11() {
  const ok = await guia({
    titulo: "Reto 2.11 — getBudgets",
    descripcion: "Suma los presupuestos (budget) de una matriz de objetos.",
    entradas: ["Array de objetos en JSON con propiedad budget"],
    ejemplo: '[{"name":"John","budget":23000},{"name":"Steve","budget":40000}] → 63000'
  });
  if (!ok) return;

  const people = await pedirJson({
    titulo: "Pega el ARRAY de OBJETOS (JSON)",
    expected: "array",
    ejemplo: [
      { name: "John", age: 21, budget: 23000 },
      { name: "Steve", age: 32, budget: 40000 },
      { name: "Martin", age: 16, budget: 2700 }
    ]
  });
  if (people === null) return;

  const budgets = people.map((p) => Number(p?.budget));
  if (budgets.some((b) => Number.isNaN(b))) {
    await Swal.fire({ icon: "error", title: "Dato inválido", text: "Cada objeto debe tener budget numérico." });
    return;
  }

  const total = budgets.reduce((acc, b) => acc + b, 0);

  await resultado({
    resHtml: `<p>Suma de budgets = <b>${total}</b></p>`,
    expHtml: `<p>Se extrajo <code>budget</code> y se sumó con <code>reduce</code>.</p>`
  });
}
```

- Explicación:

  Este ejercicio suma la propiedad budget de una lista de objetos. Se usa map() para extraer y convertir cada presupuesto con Number(p?.budget) (incluye **optional chaining** ?. para evitar errores si falta la propiedad) y luego reduce() para sumar todos los valores.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita un array de objetos en JSON.
  3. Se extrae budget con people.map((p) => Number(p?.budget)).
  4. Se valida que no existan NaN en la lista de presupuestos.
  5. Se suma con reduce((acc, b) => acc + b, 0).
  6. Se obtiene el total final.
  7. Se muestra la suma en el modal.

Reto 2 — Ejercicio 12: getStudentNames

Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 12: getStudentNames
---------------------------- */
async function reto2_ej12() {
  const ok = await guia({
    titulo: "Reto 2.12 — getStudentNames",
    descripcion: "Devuelve una matriz solo con los nombres de estudiantes.",
    entradas: ["Array de objetos en JSON con propiedad name"],
    ejemplo: '[{"name":"Steve"},{"name":"Mike"},{"name":"John"}] → ["Steve","Mike","John"]
  });
  if (!ok) return;

  const students = await pedirJson({
    titulo: "Pega el ARRAY de estudiantes (JSON)",
    expected: "array",
    ejemplo: [{ name: "Steve" }, { name: "Mike" }, { name: "John" }]
  });
  if (students === null) return;

  const names = students.map((s) => s?.name).filter((n) => typeof n === "string");

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(names))}</pre>`,
    expHtml: `<p>Se recorrió el array y se proyectó el campo <code>name</code>.</p>`
  });
}
```

- ## Explicación:

  Este ejercicio obtiene solo los nombres (name) desde un arreglo de objetos. Se aplica map() para proyectar el campo y luego filter() para quedarse únicamente con valores que realmente sean strings, evitando valores faltantes o tipos incorrectos.

- ## Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita un array de estudiantes en JSON.
  3. Se aplica map((s) => s?.name) para extraer nombres.
  4. Se filtra con filter((n) => typeof n === "string").
  5. Se obtiene el arreglo final de nombres.
  6. Se muestra el resultado en el modal.

Reto 2 — Ejercicio 13: objectToArray

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 13: objectToArray
---------------------------- */
async function reto2_ej13() {
  const ok = await guia({
    titulo: "Reto 2.13 — objectToArray",
    descripcion: "Convierte un objeto en una matriz de claves y valores.",
    entradas: ["Objeto en JSON"],
    ejemplo: '{"likes":2,"dislikes":3} → [["likes",2],["dislikes",3]]'
  });
  if (!ok) return;

  const obj = await pedirJson({
    titulo: "Pega el OBJETO JSON",
    expected: "object",
    ejemplo: { likes: 2, dislikes: 3, followers: 10 }
  });
  if (obj === null) return;

  const out = Object.entries(obj);

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Se utilizó <code>Object.entries</code> para obtener <code>[clave, valor]</code>.</p>`
  });
}
```

- Explicación:

  Este ejercicio transforma un objeto en una lista de pares [clave, valor] utilizando Object.entries(). Es equivalente al Ejercicio 10, reforzando el mismo concepto de conversión de estructuras para manipulación e iteración.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita un objeto en formato JSON.
  3. Se valida que sea un objeto válido.
  4. Se ejecuta Object.entries(obj).
  5. Se obtiene el arreglo de pares.
  6. Se presenta el resultado en el modal.

Reto 2 — Ejercicio 14: squaresSum(n)

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 14: squaresSum
---------------------------- */
async function reto2_ej14() {
  const ok = await guia({
    titulo: "Reto 2.14 — squaresSum(n)",
    descripcion: "Dado n, devuelve 1² + 2² + ... + n².",
    entradas: ["n entero (≥ 0)"],
    ejemplo: "squaresSum(3) → 14"
  });
  if (!ok) return;

  const n = await pedirNumero({ titulo: "Ingresa n (entero ≥ 0)", allowDecimal: false, allowNegative: false });
  if (n === null) return;

  let total = 0;
  for (let i = 1; i <= n; i++) total += i ** 2;

  await resultado({
    resHtml: `<p>squaresSum(${esc(n)}) = <b>${total}</b></p>`,
    expHtml: `<p>Se acumuló la suma de cuadrados iterando desde 1 hasta n.</p>`
  });
}
```

- Explicación:

  Este ejercicio calcula la suma de cuadrados desde 1 hasta n usando un bucle for. Emplea el operador de exponenciación (**) para obtener i ** 2 y acumula el total en una variable, lo cual es apropiado cuando se requiere un cálculo incremental controlado.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita n como entero mayor o igual a 0.
  3. Se inicializa total = 0.
  4. Se recorre i desde 1 hasta n con un for.
  5. En cada iteración se suma i ** 2 a total.
  6. Se obtiene el total final.
  7. Se muestra squaresSum(n) en el modal.

Reto 2 — Ejercicio 15: multiplyByLength

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 15: multiplyByLength
---------------------------- */
async function reto2_ej15() {
  const ok = await guia({
    titulo: "Reto 2.15 — multiplyByLength",
    descripcion: "Multiplica cada valor por la longitud total del array.",
    entradas: ["Lista numérica (coma o espacio)"],
    ejemplo: "[2,3,1,0] → [8,12,4,0]"
  });
  if (!ok) return;

  const nums = await pedirListaNumeros({ titulo: "Ingresa los números", placeholder: "Ej.: 2, 3, 1, 0" });
  if (!nums) return;

  const k = nums.length;
  const out = nums.map((x) => x * k);

  await resultado({
    resHtml: `<p>length = <b>${k}</b></p><pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Cada elemento se transformó como <code>x ↦ x·length</code>.</p>`
  });
}
```

- Explicación:

  Este ejercicio multiplica cada elemento del arreglo por su longitud total usando map(). Primero se obtiene k = nums.length y luego se transforma cada valor con la función flecha x => x * k, generando un nuevo arreglo sin modificar el original.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita una lista de números.
  3. Se calcula la longitud del arreglo con k = nums.length.
  4. Se aplica map((x) => x * k) para transformar cada elemento.
  5. Se obtiene el arreglo resultante out.
  6. Se muestra la longitud y el arreglo final en el modal.

Reto 2 — Ejercicio 16: countdown(n)

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 16: countdown
---------------------------- */
async function reto2_ej16() {
  const ok = await guia({
    titulo: "Reto 2.16 — countdown(n)",
    descripcion: "Devuelve un array contando desde n hasta 0.",
    entradas: ["n entero (≥ 0)"],
    ejemplo: "countdown(5) → [5,4,3,2,1,0]"
  });
  if (!ok) return;

  const n = await pedirNumero({ titulo: "Ingresa n (entero ≥ 0)", allowDecimal: false, allowNegative: false });
  if (n === null) return;

  const out = [];
  for (let i = n; i >= 0; i--) out.push(i);

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Se generó una secuencia decreciente desde n hasta 0 (incluyéndolos).</p>`
  });
}
```

- Explicación:

  Este ejercicio genera un arreglo descendente desde n hasta 0 usando un bucle for decreciente y push(). Es un caso típico de construcción incremental de arreglos cuando se necesita controlar el orden de inserción.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita n como entero mayor o igual a 0.
  3. Se inicializa out = [].
  4. Se recorre i desde n hasta 0 decrementando.
  5. En cada iteración se ejecuta out.push(i).
  6. Se obtiene el arreglo final.
  7. Se muestra out en el modal.

Reto 2 — Ejercicio 17: diffMaxMin

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 18: filterList (solo enteros)
---------------------------- */
async function reto2_ej18() {
  const ok = await guia({
    titulo: "Reto 2.18 — filterList",
    descripcion: "Filtra las cadenas de una matriz y devuelve una nueva matriz que solo contenga enteros.",
    entradas: ["Array mixto en JSON"],
    ejemplo: '[1,2,3,"x","y",10] → [1,2,3,10]'
  });
  if (!ok) return;

  const arr = await pedirJson({
    titulo: "Pega el ARRAY JSON",
    expected: "array",
    ejemplo: [1, 2, 3, "x", "y", 10]
  });
  if (arr === null) return;

  const out = arr.filter((x) => typeof x === "number" && Number.isInteger(x));

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Se conservaron únicamente valores numéricos enteros; los strings se eliminaron.</p>`
  });
}
```

- Explicación:

  Este ejercicio calcula la diferencia entre el valor máximo y mínimo de un arreglo usando Math.min(...nums) y Math.max(...nums). El spread operator (...) permite pasar el arreglo como argumentos, y luego se calcula diff = mx - mn.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita una lista de números.
  3. Se calcula el mínimo con mn = Math.min(...nums).
  4. Se calcula el máximo con mx = Math.max(...nums).
  5. Se calcula la diferencia diff = mx - mn.
  6. Se presenta el cálculo final en el modal.

Reto 2 — Ejercicio 18: filterList (solo enteros)

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 18: filterList (solo enteros)
---------------------------- */
async function reto2_ej18() {
  const ok = await guia({
    titulo: "Reto 2.18 — filterList",
    descripcion: "Filtra las cadenas de una matriz y devuelve una nueva matriz que solo contenga enteros.",
    entradas: ["Array mixto en JSON"],
    ejemplo: '[1,2,3,"x","y",10] → [1,2,3,10]'
  });
  if (!ok) return;

  const arr = await pedirJson({
    titulo: "Pega el ARRAY JSON",
    expected: "array",
    ejemplo: [1, 2, 3, "x", "y", 10]
  });
  if (arr === null) return;

  const out = arr.filter((x) => typeof x === "number" && Number.isInteger(x));

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Se conservaron únicamente valores numéricos enteros; los strings se eliminaron.</p>`
  });
}
```

- Explicación:

Este ejercicio filtra un arreglo para conservar solo números enteros. Se usa filter() con dos condiciones: typeof x === "number" para asegurar tipo numérico y Number.isInteger(x) para asegurar que no sea decimal, eliminando strings y otros tipos.

- Procedimiento:

1. Se muestra la guía del ejercicio.
2. Se solicita un arreglo mixto en formato JSON.
3. Se valida que el dato sea un array.
4. Se aplica filter((x) => typeof x === "number" && Number.isInteger(x)).
5. Se obtiene el arreglo final con enteros únicamente.
6. Se muestra el resultado en el modal.

Reto 2 — Ejercicio 19: repeat(element, times)

- ## Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 19: repeat(element, times)
---------------------------- */
async function reto2_ej19() {
  const ok = await guia({
    titulo: "Reto 2.19 — repeat(element, times)",
    descripcion: "Repite un elemento la cantidad de veces indicada y lo devuelve en un array.",
    entradas: ["Elemento (texto o número)", "times (entero ≥ 0)"],
    ejemplo: "repeat(13, 5) → [13,13,13,13,13]"
  });
  if (!ok) return;

  const tipoRes = await Swal.fire({
    title: "¿Qué tipo de elemento repetir?",
    input: "select",
    inputOptions: { number: "Número", string: "Texto" },
    showCancelButton: true,
    confirmButtonText: "Aceptar",
    cancelButtonText: "Cancelar",
    confirmButtonColor: "■#4f46e5",
    didOpen: enableEnterConfirm
  });
  if (!tipoRes.isConfirmed) return;

  let element;
  if (tipoRes.value === "number") {
    element = await pedirNumero({ titulo: "Elemento numérico", allowDecimal: true });
    if (element === null) return;
  } else {
    element = await pedirTexto({ titulo: "Elemento de texto", placeholder: "Ej.: hola" });
    if (element === null) return;
  }

  const times = await pedirNumero({ titulo: "Veces (entero ≥ 0)", allowDecimal: false, allowNegative: false });
  if (times === null) return;

  const out = Array.from({ length: times }, () => element);

  await resultado({
    resHtml: `<pre style="white-space:pre-wrap">${esc(JSON.stringify(out))}</pre>`,
    expHtml: `<p>Se creó un array de longitud <b>${times}</b> replicando el mismo elemento.</p>`
  });
}
```

- ## Explicación:

Este ejercicio crea un arreglo repitiendo un elemento una cantidad específica de veces. Utiliza Array.from() con un objeto { length: times } para generar un arreglo de tamaño definido y una función flecha que retorna el mismo elemento en cada posición. Es una forma moderna y concisa de construir arreglos repetitivos sin usar bucles explícitos.

- ## Procedimiento:

1. Se muestra la guía del ejercicio.
2. Se solicita el tipo de elemento (número o texto).
3. Se captura el valor correspondiente según el tipo elegido.
4. Se solicita times como entero mayor o igual a 0.
5. Se ejecuta Array.from({ length: times }, () => element).
6. Se genera el arreglo con el elemento repetido.
7. Se muestra el arreglo final en el modal.

Reto 2 — Ejercicio 20: vreplace (String.prototype)

- Código:

```
/* ------------------------------
   RETO 2 — EJERCICIO 20: vreplace (String.prototype)
------------------------------ */
async function reto2_ej20() {
  const ok = await guia({
    titulo: "Reto 2.20 — vreplace",
    descripcion: "Extiende el prototipo String reemplazando todas las vocales por una vocal indicada.",
    entradas: ["Texto", "Vocal de reemplazo (a,e,i,o,u)"],
    ejemplo: '"apples and bananas".vreplace("u") → "upplus und bununus"'
  });
  if (!ok) return;

  if (typeof String.prototype.vreplace !== "function") {
    // eslint-disable-next-line no-extend-native
    String.prototype.vreplace = function (vowel) {
      const v = String(vowel ?? "").toLowerCase();
      if (!/^[aeiou]$/.test(v)) return String(this);
      return String(this).replace(/[aeiouáéíóú]/gi, (m) => (m === m.toUpperCase() ? v.toUpperCase() : v));
    };
  }

  const text = await pedirTexto({ titulo: "Texto", placeholder: "Ej.: apples and bananas" });
  if (text === null) return;

  const vowel = await pedirTexto({
    titulo: "Vocal (a/e/i/o/u)",
    placeholder: "Ej.: u",
    validate: (v) => (/^[aeiou]$/i.test(v) ? undefined : "Debe ser una vocal: a, e, i, o, u.")
  });
  if (vowel === null) return;

  const out = text.vreplace(vowel);

  await resultado({
    resHtml: `<p><b>${esc(out)}</b></p>`,
    expHtml: `<p>Se reemplazaron todas las vocales por la vocal indicada, conservando mayúsculas/minúsculas.</p>`
  });
}
```

- Explicación:

  Este ejercicio extiende el prototipo String agregando un método personalizado vreplace. Se valida que no exista previamente antes de definirlo. El método usa expresiones regulares globales e insensibles a mayúsculas (/gi) para reemplazar todas las vocales por la indicada, conservando el caso original. Se utiliza String.prototype.replace() con función callback para controlar dinámicamente el reemplazo.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se verifica si String.prototype.vreplace ya existe.
  3. Si no existe, se define el nuevo método en el prototipo.
  4. Se solicita el texto a procesar.
  5. Se solicita la vocal de reemplazo con validación.
  6. Se ejecuta text.vreplace(vowel).
  7. Se muestra el texto transformado en el modal.

Reto 2 — Ejercicio 21: findNemo

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 22: capLast
---------------------------- */
async function reto2_ej22() {
  const ok = await guia({
    titulo: "Reto 2.22 — capLast",
    descripcion: "Capitaliza la última letra de cada palabra.",
    entradas: ["Texto"],
    ejemplo: 'capLast("hello world") → "hellO worlD"'
  });
  if (!ok) return;

  const text = await pedirTexto({ titulo: "Texto", placeholder: "Ej.: hello world" });
  if (text === null) return;

  const out = text
    .split(/\s+/)
    .map((w) => (w.length <= 1 ? w.toUpperCase() : w.slice(0, -1) + w.slice(-1).toUpperCase()))
    .join(" ");

  await resultado({
    resHtml: `<p><b>${esc(out)}</b></p>`,
    expHtml: `<p>Para cada palabra se transformó solo el último carácter a mayúscula.</p>`
  });
}
```

- Explicación:

  Este ejercicio busca la palabra "Nemo" dentro de una frase. Divide el texto usando split(/\s+/) para separar por espacios y aplica findIndex() para localizar la palabra ignorando mayúsculas y signos de puntuación. Si no se encuentra, retorna un mensaje alternativo.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita una frase al usuario.
  3. Se elimina espacio extra con trim().
  4. Se divide la frase en palabras con split(/\s+/).
  5. Se aplica findIndex() comparando en minúsculas.
  6. Si el índice es ≥ 0, se calcula la posición (índice + 1).
  7. Se muestra el mensaje correspondiente en el modal.

Reto 2 — Ejercicio 22: capLast

- Código:

```
/* ----------------------------
   RETO 2 — EJERCICIO 22: capLast
---------------------------- */
async function reto2_ej22() {
  const ok = await guia({
    titulo: "Reto 2.22 — capLast",
    descripcion: "Capitaliza la última letra de cada palabra.",
    entradas: ["Texto"],
    ejemplo: 'capLast("hello world") → "hellO worlD"'
  });
  if (!ok) return;

  const text = await pedirTexto({ titulo: "Texto", placeholder: "Ej.: hello world" });
  if (text === null) return;

  const out = text
    .split(/\s+/)
    .map((w) => (w.length <= 1 ? w.toUpperCase() : w.slice(0, -1) + w.slice(-1).toUpperCase()))
    .join(" ");

  await resultado({
    resHtml: `<p><b>${esc(out)}</b></p>`,
    expHtml: `<p>Para cada palabra se transformó solo el último carácter a mayúscula.</p>`
  });
}
```

- Explicación:

  Este ejercicio capitaliza la última letra de cada palabra usando split() para separar el texto, map() para transformar cada palabra y slice() para manipular cadenas. Si la palabra tiene más de un carácter, se mantiene el inicio y se convierte a mayúscula únicamente el último carácter.

- Procedimiento:

  1. Se muestra la guía del ejercicio.
  2. Se solicita un texto al usuario.
  3. Se divide el texto en palabras con split(/\s+/).
  4. Se aplica map() para transformar cada palabra.
  5. En cada palabra se usa slice(0, -1) y slice(-1).toUpperCase().
  6. Se unen las palabras con join(" ").
  7. Se muestra el texto transformado en el modal.