Día 1 · Lógica y estructuras fundamentales

Ejercicio 1: Verificador de paréntesis balanceados

Versión básica Implementa una función que reciba una cadena de símbolos `()[]{}` y devuelva `true` si están correctamente balanceados, o `false` si hay errores de cierre o anidamiento.

Versión extendida Crea un analizador que valide la jerarquía y el cierre correcto de etiquetas HTML/XML. Debe detectar errores, reportar su ubicación y generar un esquema visual del árbol de anidamiento.

Definiciones clave

Concepto	Definición
Stack (Pila)	Estructura LIFO: el último elemento en entrar es el primero en salir.
Balanceo de símbolos	Validación de apertura/cierre correcto de paréntesis, llaves o corchetes.
Parser	Algoritmo que analiza sintaxis y estructura jerárquica en cadenas como HTML.

Ejemplos

- Entrada básica: `"{[()]}"` → Salida: `true`
- Entrada básica: `"{[(])}"` → Salida: `false`
- Entrada extendida: `<div><span><p></p></span></div>` → ✓ Árbol válido
- Error extendido: `<div><span><p></span></p></div>` → ✗ Cierre incorrecto de `<span>`

Ejercicio 2: Simulador de cola en supermercado

Versión básica Diseña una clase Supermercado con métodos para agregar clientes, atenderlos en orden de llegada, y mostrar la cola actual. Utiliza una estructura FIFO.

Versión extendida Implementa una cola con prioridad que atienda clientes VIP, Express y Regular según reglas configurables. Utiliza clases personalizadas y estructuras dinámicas.

Definiciones clave

Concepto	Definición
FIFO	First In, First Out: el primero en entrar es el primero en ser atendido.
Queue (Cola)	Estructura que gestiona elementos siguiendo lógica FIFO.
Cola con prioridad	Organización de elementos según niveles de importancia, no solo por orden.

Ejemplos

- Básico:

  js

  ```
  agregarCliente("Ana");
  agregarCliente("Luis");
  atenderCliente(); // Atiende a Ana
  mostrarCola();    // ["Luis"]
  ```

- Extendido:

  js

  ```
  agregarCliente("Carlos", "VIP");
  agregarCliente("Sara", "Regular");
  agregarCliente("Elena", "Express");
  atenderCliente(); // Atiende a Carlos
  mostrarCola();    // ["Elena", "Sara"]
  ```

Ejercicio 3: Factorial iterativo y recursivo

Versión básica Crea dos funciones para calcular el factorial de un número: una usando bucle (`for`) y otra recursiva (`f(n) = n * f(n-1)`).

Versión extendida Optimiza para valores grandes con `BigInt`, mide tiempos con `console.time()` o equivalente y visualiza el árbol de llamadas recursivas.

Definiciones clave

Concepto	Definición
Factorial	Producto de enteros positivos hasta n. Ejemplo: `5! = 120`.
Iteración	Repetición mediante bucle sin llamadas internas.
Recursión	La función se llama a sí misma para resolver subproblemas.
BigInt	Tipo de dato para enteros muy grandes, como `100!`.
Benchmarking	Comparación de rendimiento entre funciones o algoritmos.

Ejemplos

- Básico:

  js

  ```
  factorialIterativo(5);   // 120
  factorialRecursivo(6);   // 720
  ```

- Extendido:

  js

  ```
  factorialBigInt(100);    // 933...000
  console.time("iterativo");
  factorialIterativo(10000);
  console.timeEnd("iterativo");
  ```