## HINTS

Encadenando un ternario:

El operador ternario es "asociativo a la derecha", lo que significa que se puede *encadenar* similar a una cadena "`if ... else if ... else if ... else`", de la siguiente manera:

```
function ejemplo() {
    return condicion1 ? valor1
         : condicion2 ? valor2
         : condicion3 ? valor3
         : valor4;
}

//  Equivale a:

function ejemplo() {
    if (condicion1) { return valor1; }
    else if (condicion2) { return valor2; }
    else if (condicion3) { return valor3; }
    else { return valor4; }
}
```

Como se puede notar, esto conforma otra manera más de simplificar nuestro código, manteniéndolo "limpio", y a la vez, conservando su funcionalidad.

### BUENAS PRÁCTICAS DE CODIFICACIÓN

De acuerdo con **The World Wide Web Consortium**, la Organización Internacional de Estándares para la **World Wide Web**, algunas de las mejores prácticas de programación al usar JavaScript, son las siguientes:

1. Haz nombres de variables y funciones fáciles, breves y legibles.
2. Evita las variables globales.
3. Comenta tanto como sea necesario, pero no más.
4. Evita mezclar con otras tecnologías.
5. Usa una notación abreviada cuando tenga sentido.
6. Modularizar: una función por tarea.
7. Evita mucho anidamiento.
8. Optimiza bucles lo más posible.

9. Agrega funcionalidad con JavaScript, no lo uses para crear demasiado contenido.
10. Construye sobre buenas ideas.

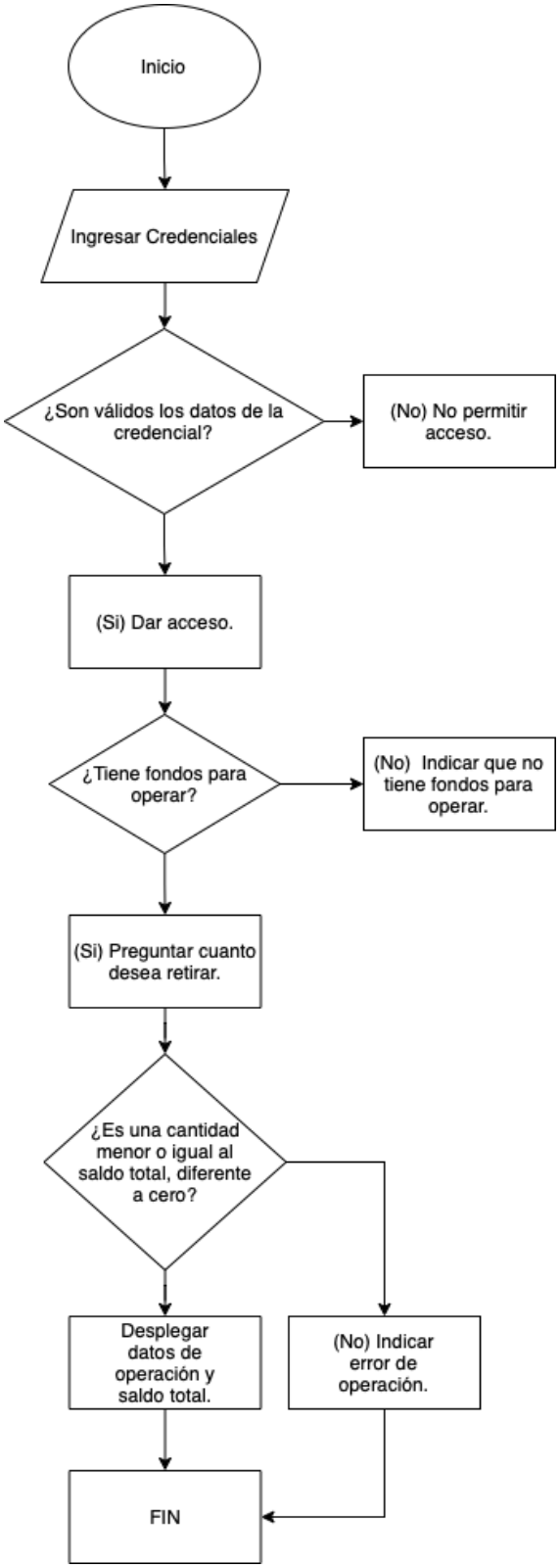
Considera que las "mejores prácticas", se refieren a lo óptimo que los desarrolladores pueden seguir al desarrollar código.

¿Por qué no establecer como objetivo implementar, al menos una de estas buenas prácticas, en tu flujo de trabajo? En el transcurso del proceso, puedes ir aplicando más de ellas para mejorar tu trabajo como desarrollador, y la calidad del código.

**IMPLEMENTANDO CÓDIGO A PARTIR DE UN DIAGRAMA DE FLUJO**

Durante este CUE se han analizado ejemplos sencillos, que permiten aprender los fundamentos del Control de Flujo, evaluando condiciones. Si se requiere abordar problemas más grandes y mucho más complejos, que necesiten condiciones de evaluación, siempre se puede considerar organizar nuestro proceso de pensamiento en un diagrama de flujo.

De hecho, éstos son una forma gráfica muy simple de representar todos los procesos necesarios para una solución informativa a un problema. Por ejemplo, si quisiéramos programar un cajero automático, se pudiese plantear en un modelo así:


Inicio
Ingresar Credenciales
¿Son válidos los datos de la credencial?
(No) No permitir acceso.
(Si) Dar acceso.
¿Tiene fondos para operar?
(No) Indicar que no tiene fondos para operar.
(Si) Preguntar cuanto desea retirar.
¿Es una cantidad menor o igual al saldo total, diferente a cero?
Desplegar datos de operación y saldo total.
(No) Indicar error de operación.
FIN

Analizando con atención este diagrama, podremos determinar cómo implementar cada proceso por sí mismo. Ahora, un ejemplo de cómo transformarlo en código, es el siguiente:

```
// Credenciales
let usuario = "alex123"
let pin = 1234

// dinero disponible en la cuenta.
let monto = 1000

function ingresarCuenta(usuario, pin) {

    // Verificacion de credenciales.
    cuenta_usuario = (usuario === "alex123" && pin === 1234) ?
'Bienvenido.' : 'Datos incorectos, intenta nuevamente.';
    console.log(cuenta_usuario)

    // Preguntamos si desea transferir fondos.
    transferir = prompt("desea realizar una transferencia? [si/no]");

    // si no tiene fondos no puede retirar dinero.
    saldo = (monto > 0) ? transferir : "No tiene fondos para operar.";

    // Si quiere retirar fondos
    if (transferir === "si") {

        // Cuanto quiere retirar?
        cuanto = prompt("Ingrese el monto que desea ingresar, no puede ser
más que $" + monto);

        // Si quiere retirar menos de lo que tiene sin que sea cero:
        if (cuanto <= monto && cuanto !== 0) {
            // mostrar datos de la transferencia.
            console.log("Monto solicitado: $" + cuanto);
            console.log("Saldo total: $" + (monto - cuanto));
        } else {
            // si pide mas de lo que tiene, error!
            console.log("No puede solicitar ese monto.");
        }
    } else {
        // NO quiere retirar fondos
        console.log("ok, cerrando cuenta.");
    }
}

```

```
// Llamamos a la funcion, iniciamos cajero.
ingresarCuenta(usuario, pin);
```

Se recomienda copiar el código, e interactuar con él, para revisar si hace lo que propone el diagrama de flujo. Si no, puedes intentar crear una propia implementación de esta solución, y así mejorarás tus habilidades para diseñar programas.