BOTIC-SOFOF-24-28-13-0081

Módulo 3: Fundamentos de Programación en JavaScript (60 Horas) / ELE069950089611

AE2_ABPRO-Ejercicio grupal [Actividad Evaluada]

AE2_ABPRO-Ejercicio grupal [Actividad Evaluada]

Contexto

Un supermercado quiere implementar un sistema básico en JavaScript que registre las compras de los clientes en una terminal de autoservicio. El sistema debe almacenar información del cliente y los productos que va añadiendo al carrito, calcular el total, aplicar descuentos si se superan ciertos montos, y validar condiciones como carritos vacíos o valores inválidos. El objetivo es que el equipo diseñe la lógica utilizando variables correctamente y domine los operadores, tipos de datos y control de flujo.

Actividad

Trabajen en equipo para desarrollar una solución en JavaScript que cubra los siguientes aspectos:

- Declaren variables para:
  - Nombre del cliente.
  - Lista de productos (usar un arreglo de objetos con nombre, cantidad y precio).
  - Total de compra.
  - Descuento aplicado (si corresponde).

- Apliquen nombres significativos a todas las variables del sistema.
- Utilicen constantes para definir límites como monto mínimo para aplicar descuento o máximo de productos permitidos.
- Manipulen tipos simples (string, number, boolean) y complejos (arreglos y objetos) según corresponda.
- Controlen el scope de las variables según el diseño modular del programa (funciones).
- Usen operadores aritméticos para calcular subtotales y totales.
- Consideren la precedencia de operadores al aplicar fórmulas (como descuento o impuestos).
- Usen operadores de comparación y lógicos para validar:
  - Si el carrito está vacío.
  - Si el monto total supera un cierto límite.
  - Si algún precio o cantidad ingresada es inválida.

- Simulen acciones del cliente como añadir productos, quitar productos y finalizar compra.
- Usen if y else para mostrar diferentes mensajes según el total (por ejemplo: “gracias por tu compra”, “tu carrito está vacío”, “aplicaste un descuento”).
- Implementen un diagrama de flujo simple que explique la lógica principal del sistema y represente visualmente cómo se toman decisiones.
- Prueben diferentes condiciones de borde:
  - Producto con cantidad 0.
  - Total de compra en 0.
  - Lista de productos vacía.
  - Precio negativo.

Al final, el equipo debe presentar su solución y explicar brevemente cómo resolvieron los siguientes puntos:

- ¿Cómo evitaron errores por mal uso de variables?
- ¿Qué beneficios vieron en el uso de objetos para representar productos?
- ¿Cómo manejaron una lógica clara y eficiente con if/else?
- ¿Qué condiciones de borde encontraron más retadoras?

Entrega:

- Archivo zip con tu código y además la reflexión.
- Ejecución: Grupal.
- Duración: 1 jornada de clases.

Estado de la entrega

Estado de la entrega	Todavía no se han realizado envíos
Estado de la calificación	Sin calificar
Última modificación	-

DESARROLLO EJERCICIO - SALA 2

a) PREGUNTAS.

1. ¿Cómo evitaron errores por mal uso de variables?

   El código desarrollado evita errores principalmente al usar:
   - Constantes bien definidas (DESCUENTO_POR_MONTO, MAX_PRODUCTOS) para valores que no cambian, lo que previene errores de reescritura.
   - Validaciones tempranas en la función agregarProducto() que validan si la cantidad <= 0 o precio < 0, impidiendo de esta forma que se agreguen productos inválidos.
   - Uso adecuado del scope (let, const) para mantener controladas las variables y no generar conflictos.

2. ¿Qué beneficios vieron en el uso de objetos para representar productos?

   Usar objetos para los productos ({nombre, cantidad, precio}) permitió:
   - Agrupar todos los datos de un producto en una sola estructura.
   - Acceder fácilmente a cada propiedad dentro de bucles o funciones (p.nombre, p.precio, etc.).
   - Facilitar operaciones como forEach o findIndex, manteniendo el código limpio y fácil de mantener.

3. ¿Cómo manejaron una lógica clara y eficiente con if/else?

   Se aplicaron condiciones if/else de forma clara:
   - En agregarProducto(), se manejan casos inválidos antes de continuar con el flujo normal (validación temprana).
   - En calcularTotal(), se usa un if para aplicar el descuento solo si corresponde, sin lógica innecesaria.
   - En finalizarCompra(), se verifica si el carrito está vacío antes de continuar, evitando errores al mostrar resultados.

   Esto permite que la lógica del programa fluya sin confusión, actuando solo cuando es necesario.

4. ¿Qué condiciones de borde encontraron más retadoras?

   Algunas condiciones desafiantes que el código maneja son:
   - Productos con cantidad cero o precio negativo, que podrían romper los cálculos o mostrar datos inválidos. Esto se controla correctamente con una validación.
   - Intentar eliminar un producto que no existe: se controla con findIndex y un mensaje claro si el producto no se encuentra.
   - Carrito vacío al finalizar la compra, que se maneja con un mensaje de advertencia sin continuar el proceso.
   - Limitar la cantidad de productos (MAX_PRODUCTOS), evitando que se sobrecargue el carrito.

b) CODIGO A PRESENTAR SCRIPT.JS

```
const DESCUENTO_POR_MONTO = 100;
const PORCENTAJE_DESCUENTO = 0.1;
const MAX_PRODUCTOS = 10;

let nombreCliente = "Linn";
let carrito = [];
let totalCompra = 0;
let descuentoAplicado = false;

function agregarProducto(nombre, cantidad, precio) {
	if (cantidad <= 0 || precio < 0) {
		console.log(`□ Producto inválido: "${nombre}". Cantidad o precio incorrecto.`);
		return;
	}

	if (carrito.length >= MAX_PRODUCTOS) {
		console.log(`□ No puedes agregar más de ${MAX_PRODUCTOS} productos.`);
		return;
	}

	carrito.push({ nombre, cantidad, precio });
	console.log(`□ Producto añadido: ${cantidad} x ${nombre} ($${precio} c/u)`);
}

function quitarProducto(nombre) {
	const index = carrito.findIndex(p => p.nombre === nombre);
	if (index !== -1) {
		carrito.splice(index, 1);
		console.log(`🗑□ Producto eliminado: ${nombre}`);
	} else {
		console.log(`□ Producto no encontrado: ${nombre}`);
	}
}

function calcularTotal() {
	totalCompra = 0;
	carrito.forEach(p => {
		totalCompra += p.precio * p.cantidad;
	});

	if (totalCompra >= DESCUENTO_POR_MONTO) {
		const descuento = totalCompra * PORCENTAJE_DESCUENTO;
		totalCompra -= descuento;
		descuentoAplicado = true;
	} else {
		descuentoAplicado = false;
	}
}
```

```
function finalizarCompra() {
	if (carrito.length === 0) {
		console.log("🛒 Tu carrito está vacío.");
		return;
	}

	calcularTotal();

	console.log(`🧾 Cliente: ${nombreCliente}`);
	console.log("📦 Productos:");
	carrito.forEach(p => {
		console.log(` - ${p.cantidad} x ${p.nombre} ($${p.precio})`);
	});

	console.log(`💵 Total: $${totalCompra.toFixed(2)}`);

	if (descuentoAplicado) {
		console.log("🎉 ¡Descuento aplicado del 10%!");
	} else {
		console.log("ℹ️ No se aplicó descuento.");
	}

	console.log("🙏 Gracias por tu compra.");
}
```

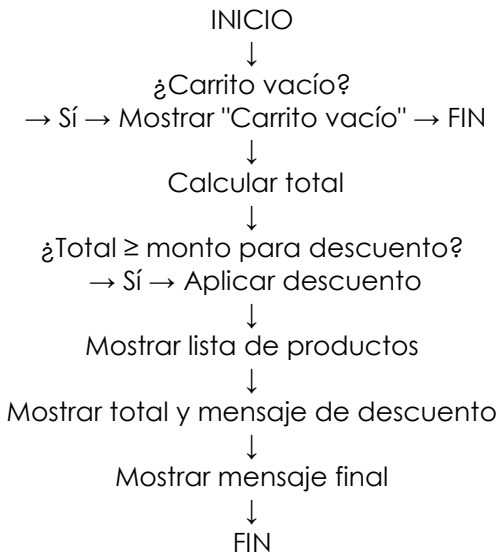
Pruebas realizadas:

```
agregarProducto("Pan", 2, 1.5);
agregarProducto("Leche", 1, 2.2);
agregarProducto("Huevos", 0, 3); // inválido
agregarProducto("Arroz", 1, -5); // inválido
agregarProducto("Café", 1, 98);

quitarProducto("Arroz"); // no existe
quitarProducto("Pan");

finalizarCompra();
```

c) DIAGRAMA DE FLUJO SIMPLIFICADO:



d) OTROS CÓDIGOS:

Adicionalmente se adjunta códigos realizados que fueron comparados en cuanto a funcionamiento, estructura y logia de programación con funciones y sin funciones, cumpliendo el mismo objetivo.