## Desafío en clase: "Gestor de Tareas en Elixir"

### Objetivo

Desarrollar una aplicación sencilla de línea de comandos en Elixir que funcione como un gestor de tareas. Esta aplicación permitirá agregar, listar y marcar tareas como completadas, utilizando y demostrando conocimiento de los operadores básicos, tipos de datos, funciones y módulos en Elixir.

### Requerimientos

1. **Crear un módulo `TaskManager`**:
    - Define un módulo `TaskManager` que contendrá todas las funciones necesarias.
2. **Estructuras de Datos**:
    - Usa una lista de mapas para almacenar las tareas. Cada tarea será un mapa con las siguientes claves: `:id` (entero), `:description` (cadena de texto) y `:completed` (booleano).
3. **Funciones Básicas**:
    - `add_task(tasks, description)`: Agrega una nueva tarea a la lista de tareas. Asigna automáticamente un `id` único a cada tarea.
    - `list_tasks(tasks)`: Muestra todas las tareas con su `id`, `description` y estado `completed`.
    - `complete_task(tasks, id)`: Marca la tarea con el `id` dado como completada.
4. **Operadores y Estructuras de Control**:
    - Utiliza operadores matemáticos y de comparación para gestionar los `id` de las tareas.
    - Usa declaraciones `if` para manejar las operaciones de marcado de tareas como completadas.
5. **Interacción con el Usuario**:
    - Usa el módulo `IO` para interactuar con el usuario, permitiendo agregar, listar y completar tareas.
6. **Funcionalidad Completa**:
    - Crea una función `run` en el módulo `TaskManager` que ejecute un bucle de interacción con el usuario, permitiéndole seleccionar opciones hasta que decida salir.

## Desafío autonomo: "Gestor de Inventario en Elixir"

### Objetivo

Desarrollar una aplicación sencilla de línea de comandos en Elixir que funcione como un gestor de inventario de productos. Esta aplicación permitirá agregar productos, vender productos, aumentar el stock, generar un carrito de compras, y generar venta final con los productos de carrito.

### Requerimientos

1. **Crear un módulo `InventoryManager`**:
   - Define un módulo `InventoryManager` que contendrá todas las funciones necesarias.
2. **Estructuras de Datos**:
   - Usa una lista de mapas para almacenar los productos. Cada producto será un mapa con las siguientes claves: `:id` (entero), `:name` (cadena de texto), `:price` (decimal), y `:stock` (entero).
   - Usa una lista de tuplas para representar el carrito de compras. Cada tupla contendrá el `:id` del producto y la `:quantity` (cantidad).
3. **Funciones Básicas**:
   - `add_product(inventory, name, price, stock)`: Agrega un nuevo producto al inventario. Asigna automáticamente un `id` único a cada producto.
   - `list_products(inventory)`: Muestra todos los productos con su `id`, `name`, `price` y `stock`.
   - `increase_stock(inventory, id, quantity)`: Aumenta el stock de un producto existente.
   - `sell_product(inventory, id, quantity)`: Reduce el stock de un producto y agrega el producto al carrito de compras.
   - `view_cart(cart)`: Muestra los productos en el carrito de compras con sus cantidades y el costo total.
   - `checkout(inventory, cart)`: Realiza el cobro de los productos en el carrito y vacía el carrito.
4. **Operadores y Estructuras de Control**:
   - Utiliza operadores matemáticos y de comparación para gestionar los `id`, `price` y `stock` de los productos.
   - Usa declaraciones `if` para manejar las operaciones de venta de productos y verificar la disponibilidad de stock.
5. **Interacción con el Usuario**:
   - Usa el módulo `IO` para interactuar con el usuario, permitiendo agregar productos, listar productos, aumentar stock, vender productos, ver el carrito de compras y realizar el checkout.
6. **Funcionalidad Completa**:

- Crea una función `run` en el módulo `InventoryManager` que ejecute un bucle de interacción con el usuario, permitiéndole seleccionar opciones hasta que decida salir.