Repaso Python Temas 1 – 5

Ejercicio 2 - Sistema de Inventario con Persistencia en JSON

Desarrolla un sistema de inventario para una tienda que gestione productos y permita guardar/cargar datos desde archivos JSON. El sistema debe incluir control de stock, registro de movimientos y persistencia de datos.

Clase Producto

Atributos de instancia:

- codigo (str): Código único del producto (ej: "PROD001")
- nombre (str): Nombre del producto
- categoria (str): Categoría del producto (ej: "Electrónica", "Alimentación")
- precio (float): Precio unitario del producto
- stock (int): Cantidad disponible en inventario
- fecha_creacion (datetime): Fecha de creación del producto en el sistema

Métodos de instancia:

__init__(self, codigo: str, nombre: str, categoria: str, precio: float, stock: int, fecha_creacion: Optional[datetime] = None) -> None

* Entrada: Todos los atributos del producto
* Validaciones:
  - precio debe ser mayor que 0, si no lanzar ValueError con mensaje: "El precio debe ser mayor que 0"
  - stock debe ser mayor o igual a 0, si no lanzar ValueError con mensaje: "El stock no puede ser negativo"
* Lógica: Si fecha_creacion es None, usar datetime.now()

agregar_stock(self, cantidad: int) -> None

* Entrada: cantidad a agregar (int)
* Validación: Si cantidad <= 0, lanzar ValueError con mensaje: "La cantidad debe ser positiva"
* Acción: Incrementar el atributo stock con la cantidad

reducir_stock(self, cantidad: int) -> bool

* Entrada: cantidad a reducir (int)
* Validaciones:
  - Si cantidad <= 0, lanzar ValueError con mensaje: "La cantidad debe ser positiva"
  - Si cantidad > stock, retornar False (no hay suficiente stock)
* Acción: Si hay stock suficiente, reducir el atributo stock y retornar True
* Salida: True si se pudo reducir, False si no había stock suficiente

calcular_valor_inventario(self) -> float

* Entrada: Ninguna
* Salida: Valor total del stock de este producto (stock * precio)

to_dict(self) -> dict

* Entrada: Ninguna

* Salida: Diccionario con todos los atributos del producto:

```
{
    'codigo': str,
    'nombre': str,
    'categoria': str,
    'precio': float,
    'stock': int,
    'fecha_creacion': str  # formato ISO: fecha_creacion.isoformat()
}
```

__str__(self) -> str
* Salida: String con formato:
"[PROD001] Laptop Dell - Electrónica | Precio: 899.99€ | Stock: 15 unidades"

Método de clase (estático):
@staticmethod from_dict(data: dict) -> 'Producto'
* Entrada: Diccionario con la estructura del método to_dict()
* Salida: Nueva instancia de Producto creada a partir del diccionario
* Lógica: Convertir el string fecha_creacion de vuelta a datetime usando datetime.fromisoformat()

Clase MovimientoInventario
Atributos de instancia:

- producto_codigo (str): Código del producto afectado
- tipo (str): Tipo de movimiento - valores permitidos: "ENTRADA" o "SALIDA"
- cantidad (int): Cantidad del movimiento
- fecha (datetime): Fecha del movimiento
- observaciones (str): Notas adicionales sobre el movimiento

Métodos de instancia:
__init__(self, producto_codigo: str, tipo: str, cantidad: int, observaciones: str = "", fecha: Optional[datetime] = None) -> None

- Entrada: Todos los parámetros
- Validaciones:
    - “tipo” debe ser "ENTRADA" o "SALIDA", si no lanzar ValueError con mensaje: "El tipo debe ser 'ENTRADA' o 'SALIDA'"
    - “cantidad” debe ser mayor que 0, si no lanzar ValueError con mensaje: "La cantidad debe ser positiva"

* Lógica: Si fecha es None, usar datetime.now()

to_dict(self) -> dict
* Entrada: Ninguna
* Salida: Diccionario con todos los atributos:

```
{
    'producto_codigo': str,
    'tipo': str,
    'cantidad': int,
    'fecha': str,  # formato ISO
```

```
    'observaciones': str
}
```

__str__(self) -> str
* Salida: String con formato:
"[02/02/2026 14:30] ENTRADA - PROD001 - 10 unidades - Reposición mensual"

Método de clase (estático):
@staticmethod from_dict(data: dict) -> 'MovimientoInventario'
* Entrada: Diccionario con la estructura del método to_dict()
* Salida: Nueva instancia de MovimientoInventario

Clase Inventario
Atributos de instancia:
- productos (dict[str, Producto]): Diccionario donde la clave es el código del producto y el valor es la instancia de Producto
- movimientos (list[MovimientoInventario]): Lista con todos los movimientos registrados
- archivo_productos (str): Ruta del archivo JSON para guardar productos
- archivo_movimientos (str): Ruta del archivo JSON para guardar movimientos

Métodos de instancia:
__init__(self, archivo_productos: str = "productos.json", archivo_movimientos: str = "movimientos.json") -> None
* Entrada: Rutas de los archivos JSON (con valores por defecto)
* Inicializar:
  - productos como diccionario vacío
  - movimientos como lista vacía
  - Guardar las rutas en los atributos correspondientes

agregar_producto(self, producto: Producto) -> bool
* Entrada: Instancia de Producto
* Validación: Si ya existe un producto con ese código en el diccionario productos, imprimir mensaje de error y retornar False
* Acción: Si no existe, añadir al diccionario usando el código como clave
* Salida: True si se agregó, False si ya existía

buscar_producto(self, codigo: str) -> Optional[Producto]
* Entrada: Código del producto
* Salida: Instancia de Producto si existe, None si no se encuentra

registrar_entrada(self, codigo: str, cantidad: int, observaciones: str = "") -> bool
* Entrada: Código del producto, cantidad y observaciones opcionales
* Lógica:
  - Buscar el producto usando buscar_producto()
  - Si no existe, imprimir error y retornar False
  - Crear un MovimientoInventario con tipo "ENTRADA"

- Llamar al método agregar_stock() del producto
- Añadir el movimiento a la lista movimientos
- Imprimir mensaje de éxito

* Manejo de errores: Capturar ValueError del MovimientoInventario o del agregar_stock y mostrar el mensaje
* Salida: True si se registró correctamente, False en caso contrario

registrar_salida(self, codigo: str, cantidad: int, observaciones: str = "") -> bool
* Entrada: Código del producto, cantidad y observaciones opcionales
* Lógica:
  - Buscar el producto
  - Si no existe, imprimir error y retornar False
  - Intentar reducir el stock usando reducir_stock()
  - Si retorna False, imprimir "Stock insuficiente" y retornar False
  - Si retorna True, crear un MovimientoInventario con tipo "SALIDA"
  - Añadir el movimiento a la lista movimientos
  - Imprimir mensaje de éxito
* Manejo de errores: Capturar ValueError y mostrar el mensaje
* Salida: True si se registró correctamente, False en caso contrario

listar_productos(self, categoria: Optional[str] = None) -> None
* Entrada: categoria opcional para filtrar
* Salida: Ninguna (imprime por pantalla)
* Lógica:
  - Si no hay productos, imprimir "No hay productos en el inventario"
  - Si se proporciona categoria, filtrar solo productos de esa categoría
  - Mostrar tabla formateada con: Código, Nombre, Categoría, Precio, Stock, Valor Total
  - Al final, mostrar el valor total del inventario (suma de todos los calcular_valor_inventario())

listar_movimientos(self, codigo_producto: Optional[str] = None, limite: int = 10) -> None
* Entrada:
  - codigo_producto: Opcional, para filtrar movimientos de un producto específico
  - limite: Número máximo de movimientos a mostrar (por defecto 10)
* Salida: Ninguna (imprime por pantalla)
* Lógica:
  - Si no hay movimientos, imprimir "No hay movimientos registrados"
  - Filtrar por codigo_producto si se proporciona
  - Ordenar movimientos por fecha (más recientes primero)
  - Mostrar solo los últimos limite movimientos
  - Usar el método __str__() de cada movimiento

guardar_datos(self) -> bool
* Entrada: Ninguna
* Salida: True si se guardó correctamente, False si hubo error
* Lógica:
    - Convertir todos los productos a diccionarios usando to_dict()
    - Guardar la lista de diccionarios en archivo_productos usando json.dump() con indent=4 y ensure_ascii=False
    - Convertir todos los movimientos a diccionarios usando to_dict()
    - Guardar la lista de diccionarios en archivo_movimientos
* Manejo de errores: Usar try-except para capturar excepciones de escritura, imprimir el error y retornar False

cargar_datos(self) -> bool
* Entrada: Ninguna
* Salida: True si se cargó correctamente, False si hubo error o no existen los archivos
* Lógica:
    - Verificar si los archivos existen usando os.path.exists() (importar os)
    - Si no existen, imprimir mensaje informativo y retornar False
    - Cargar los datos de productos usando json.load()
    - Convertir cada diccionario a Producto usando Producto.from_dict()
    - Almacenar en el diccionario productos usando el código como clave
    - Hacer lo mismo con los movimientos
* Manejo de errores: Usar try-except para capturar excepciones, imprimir el error y retornar False

generar_informe_categoria(self) -> dict[str, dict]
* Entrada: Ninguna
* Salida: Diccionario donde cada clave es una categoría y el valor es otro diccionario:

```
{
    'Electrónica': {
        'productos': int,      # cantidad de productos en esta categoría
        'stock_total': int,    # suma de stock de todos los productos
        'valor_total': float   # suma de valor_inventario de todos los
productos
    },
    'Alimentación': { ... }
}
```