1. Estructuras de control utilizadas

En el desarrollo del sistema de gestión de almacén se han empleado diversas estructuras de control propias del lenguaje Python para garantizar un flujo de ejecución correcto y robusto.

Condicionales (`if`, `elif`, `else`)

Se utilizan para la validación de datos, la toma de decisiones y el control del comportamiento del programa según el estado de la aplicación.

Algunos usos destacados son:

- Verificación de que los datos introducidos por el usuario sean válidos (precio positivo, cantidades no negativas, porcentajes dentro de rango).
- Comprobación de la existencia de productos antes de realizar operaciones como venta, eliminación o aplicación de descuentos.
- Selección del mensaje adecuado según el resultado de cada operación.

Estas estructuras permiten evitar estados inconsistentes y mejorar la experiencia de usuario.

Bucles (`for`)

Los bucles se utilizan para recorrer colecciones de datos, principalmente listas de productos.

Se emplean para:

- Listar todos los productos del almacén.
- Generar informes detallados.
- Cargar productos desde el archivo JSON al iniciar la aplicación.

Comprensiones de listas

Se utilizan comprensiones de listas para realizar búsquedas y filtrados de forma más concisa y legible, como en la búsqueda de productos por nombre.

Este enfoque reduce la complejidad del código y mejora su claridad.

Manejo de excepciones (`try-except`)

El programa utiliza bloques `try-except` para controlar errores que pueden producirse por entradas incorrectas del usuario o problemas de lectura y escritura de archivos.

Gracias a este mecanismo:

- La aplicación no se cierra de forma inesperada.
- Se muestran mensajes de error claros al usuario.
- Se garantiza un funcionamiento más estable.

---

2. Decisiones de diseño tomadas

El diseño del programa sigue principios sólidos de organización y mantenimiento del código.

Separación de responsabilidades

Se ha dividido la aplicación en tres bloques principales:

- **Modelo**: la clase `Producto`, que representa los datos de un producto.
- **Lógica de negocio**: la clase `Almacen`, que gestiona todas las operaciones sobre los productos.
- **Interfaz gráfica**: la clase `InterfazAlmacen`, que se encarga de la interacción con el usuario.

Esta separación facilita la comprensión del código y permite realizar modificaciones sin afectar a otras partes del sistema.

Programación Orientada a Objetos

El uso de clases y métodos permite encapsular los datos y comportamientos relacionados, mejorando la reutilización del código y su escalabilidad.

Cada clase tiene una responsabilidad clara y definida.

Persistencia de datos con JSON

Se ha optado por un archivo JSON para almacenar la información del almacén debido a su simplicidad y facilidad de uso.

El uso de métodos como `to_dict` y `from_dict` permite convertir los objetos en estructuras serializables y recuperarlos correctamente al reiniciar la aplicación.

Centralización de estilos

Los colores y fuentes se almacenan en diccionarios globales, lo que permite mantener coherencia visual y simplifica futuras modificaciones de la interfaz.

---

3. Pruebas realizadas

Las pruebas realizadas han sido principalmente manuales, aprovechando la interfaz gráfica del programa.

Pruebas funcionales

- Creación de productos con datos válidos e inválidos.
- Venta de productos con stock suficiente e insuficiente.
- Modificación del stock con valores positivos y negativos.
- Aplicación y eliminación de descuentos.

- Búsqueda y eliminación de productos.

Pruebas de validación de errores

- Introducción de valores no numéricos en campos numéricos.
- Uso de identificadores inexistentes.
- Introducción de porcentajes fuera de rango.

En todos los casos, el programa responde correctamente mostrando mensajes adecuados sin provocar errores críticos.

Pruebas de persistencia

Se comprobó que los datos se mantienen correctamente al cerrar y volver a abrir la aplicación, verificando la correcta lectura y escritura del archivo JSON.

---

4. Posibles mejoras y estructuras de Python para implementarlas

Uso de una base de datos

Como mejora futura, se podría sustituir el archivo JSON por una base de datos SQLite para gestionar un mayor volumen de datos.

Estructuras implicadas:

- Módulo `sqlite3`
- Consultas SQL para operaciones CRUD

Implementación de pruebas automáticas

Se podrían añadir pruebas automáticas para la lógica del almacén utilizando el módulo `unittest` o bibliotecas como `pytest`.

Esto permitiría detectar errores de forma temprana y mejorar la calidad del software.

Exportación de informes

Otra mejora posible es la exportación de informes a formatos como CSV o PDF.

Estructuras utilizadas:

- Módulo `csv`
- Librerías externas para generación de documentos

Optimización de búsquedas

Actualmente los productos se almacenan en una lista. Para mejorar el rendimiento, se podría usar un diccionario indexado por ID.

```
self.productos = {id: producto}
```

Esto permitiría búsquedas más eficientes.

Conclusión

El sistema implementa correctamente las estructuras de control fundamentales de Python, aplica principios de programación orientada a objetos y presenta una separación clara entre lógica y presentación. Además, gestiona errores y persistencia de datos de forma adecuada, lo que da como resultado una aplicación funcional, robusta y fácilmente ampliable en futuras versiones.