Manejo de Archivos TXT y CSV en Python

¿Por qué necesitamos guardar datos?

Cuando escribimos un programa, la información que procesamos existe solo mientras el programa está en ejecución. Si cerramos la aplicación, toda esa información desaparece. La **persistencia de datos** nos permite almacenar información de forma permanente en archivos, para poder recuperarla y usarla en futuras ejecuciones.

Los archivos son la forma más básica y accesible de guardar información. Python nos proporciona herramientas sencillas y potentes para trabajar con diferentes tipos de archivos, permitiéndonos crear aplicaciones que mantienen su estado entre ejecuciones.

Sistema de notas

Guardar calificaciones de estudiantes

Registro de compras

Historial de transacciones

Lista de tareas

Tareas pendientes y completadas

Archivos TXT: Almacenamiento de Texto Plano

Los archivos de texto (.txt) son el formato más simple para almacenar información. Guardan datos como texto plano, línea por línea, sin ninguna estructura predefinida. Son perfectos cuando necesitas almacenar información sencilla como registros, logs, o notas sin una estructura compleja.

Python hace que trabajar con estos archivos sea extremadamente sencillo gracias a la función `open()` y el gestor de contexto `with`, que se encarga automáticamente de cerrar el archivo después de usarlo, incluso si ocurre un error.

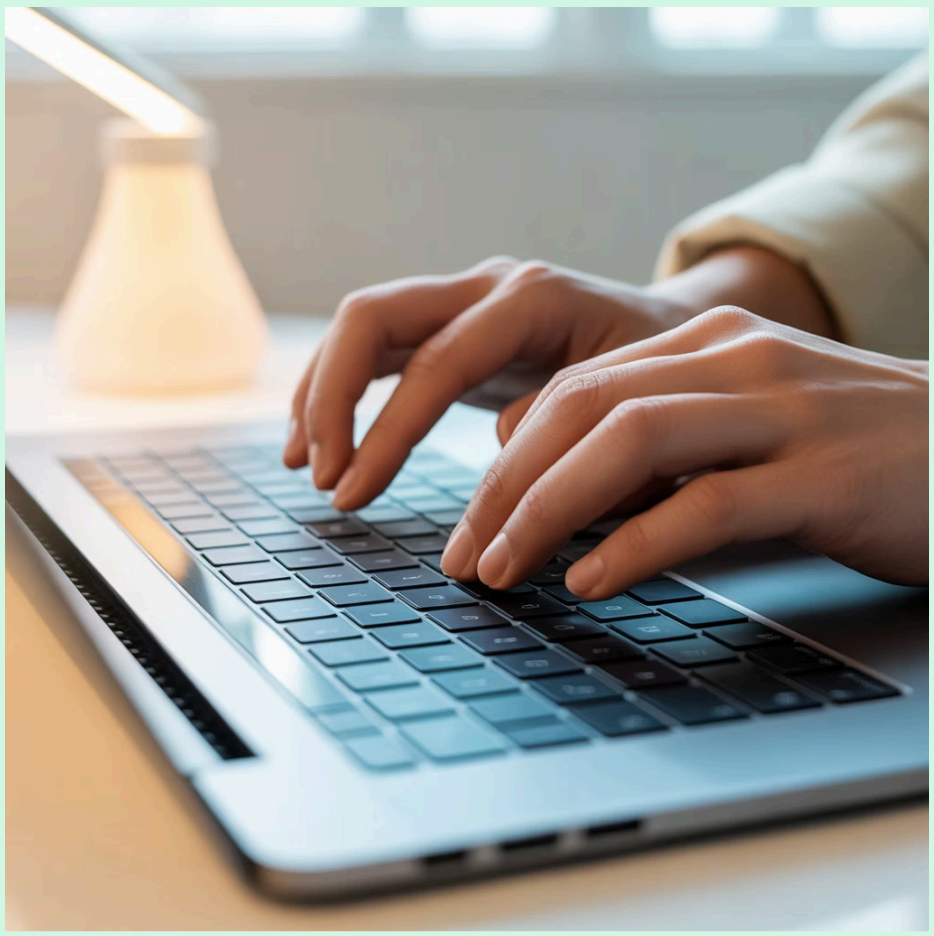
Ventajas

- Fáciles de crear y leer
- Compatibles con cualquier editor
- Perfectos para datos simples

Casos de uso

- Registros de eventos (logs)
- Configuraciones básicas
- Mensajes y notas

Escribir en un Archivo TXT



Para crear o escribir en un archivo de texto, utilizamos la función `open()` con el modo `'w'` (write). Este modo tiene un comportamiento importante: si el archivo no existe, lo crea automáticamente; si ya existe, **sobrescribe** todo su contenido anterior.

El bloque `with` es una buena práctica porque garantiza que el archivo se cierre correctamente, liberando recursos del sistema incluso si ocurre un error durante la escritura.

```
# Creamos o abrimos un archivo en modo escritura ('w')
with open("notas.txt", "w") as archivo:
    archivo.write("Nombre: Ana - Nota: 9\n")
    archivo.write("Nombre: Luis - Nota: 8\n")
    archivo.write("Nombre: Carla - Nota: 10\n")

print("Archivo creado y datos guardados con éxito.")
```

01

Modo 'w' sobrescribe

Crea el archivo nuevo o borra el contenido existente

02

Saltos de línea

Usamos \n para separar cada línea de texto

03

Cierre automático

El bloque with cierra el archivo al terminar

Leer Archivos TXT: Dos Métodos

Método 1: Lectura completa

El método `read()` carga todo el contenido del archivo en una sola cadena de texto. Es útil cuando el archivo no es muy grande y quieres procesar todo su contenido de una vez.

```
with open("notas.txt", "r") as archivo:
    contenido = archivo.read()
    print("Contenido del archivo:")
    print(contenido)
```

Este método es simple pero puede consumir mucha memoria si el archivo es muy grande.

Método 2: Lectura línea por línea

Iterar sobre el archivo directamente es más eficiente con la memoria, especialmente para archivos grandes. Python lee una línea a la vez, procesándola antes de pasar a la siguiente.

```
with open("notas.txt", "r") as archivo:
    for linea in archivo:
        # strip() elimina espacios y saltos
        print(linea.strip())
```

El método `strip()` elimina espacios en blanco y saltos de línea al inicio y final de cada línea.

Agregar Contenido sin Borrar

Archivo existente

Contiene datos previos que queremos conservar

Modo 'a' (append)

Abre el archivo para añadir al final

Datos preservados

Nuevo contenido se agrega sin borrar el anterior

```
with open("notas.txt", "a") as archivo:
    archivo.write("Nombre: Juan - Nota: 7\n")
```

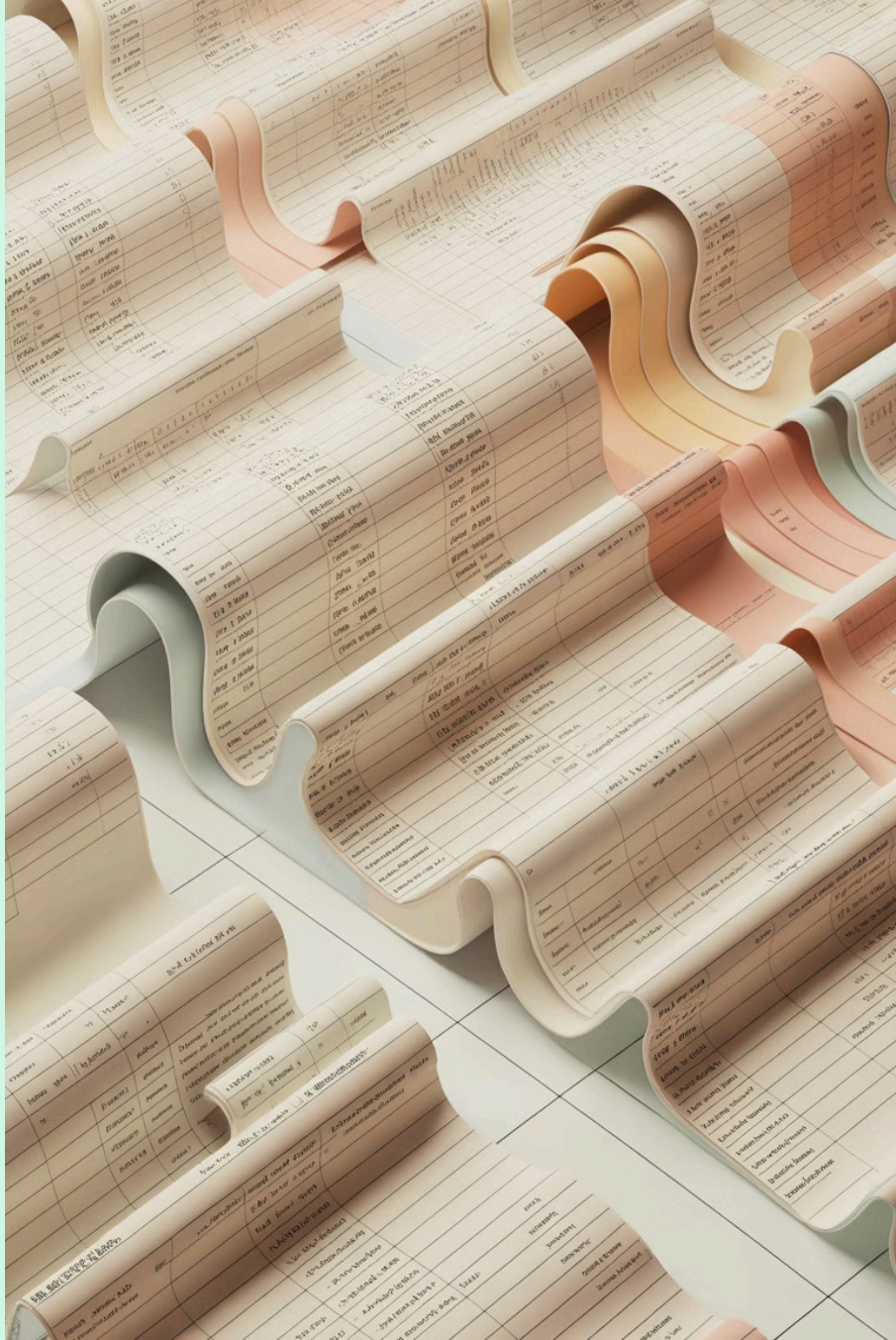
El modo `'a'` (append) es perfecto para logs o registros donde necesitas ir agregando información continuamente sin perder el historial previo. Si el archivo no existe, se crea automáticamente, igual que con el modo `'w'`.

> 💡 **Consejo:** Usa el modo 'a' cuando necesites mantener un historial o log de eventos. Es ideal para sistemas de registro donde cada ejecución del programa debe agregar nuevas entradas sin eliminar las anteriores.

Archivos CSV

Datos estructurados en formato tabla

Los archivos CSV (Comma Separated Values, o valores separados por comas) son uno de los formatos más utilizados para almacenar datos estructurados. Funcionan como una tabla o una hoja de cálculo, donde cada fila representa un registro y las columnas se separan mediante comas.



¿Qué es un Archivo CSV?

Un archivo CSV es esencialmente una tabla guardada como texto plano. La primera fila suele contener los nombres de las columnas (encabezados), y cada fila siguiente representa un registro con valores separados por comas.

Ejemplo de estructura:

```
nombre,nota
Ana,9
Luis,8
Carla,10
```

Estructura tabular

Datos organizados en filas y columnas

Compatible con Excel

Se puede abrir en hojas de cálculo

Fácil de procesar

Ideal para análisis de datos

Escribir un Archivo CSV

Python incluye un módulo especializado llamado `csv` que facilita enormemente el trabajo con este tipo de archivos. Este módulo se encarga de todos los detalles técnicos, como el formato correcto de las comas y el manejo de textos que contienen comas dentro de los valores.

```
import csv

with open("alumnos.csv", "w", newline="") as archivo:
    escritor = csv.writer(archivo)
    escritor.writerow(["nombre", "nota"])  # encabezado
    escritor.writerow(["Ana", 9])
    escritor.writerow(["Luis", 8])
    escritor.writerow(["Carla", 10])

print("Archivo CSV creado correctamente.")
```

01

Importar módulo csv

Acceso a funciones especializadas

02

Crear escritor

csv.writer() prepara el archivo

03

Escribir filas

writerow() agrega cada registro

⚠️ **Importante:** El parámetro `newline=""` evita que se agreguen líneas vacías entre las filas en algunos sistemas operativos.

Leer un Archivo CSV

Leer un archivo CSV es tan sencillo como escribirlo. El módulo `csv` se encarga de dividir cada línea en valores separados, entregándote cada fila como una lista de Python. Esto hace que sea muy fácil procesar los datos usando bucles y estructuras de control.

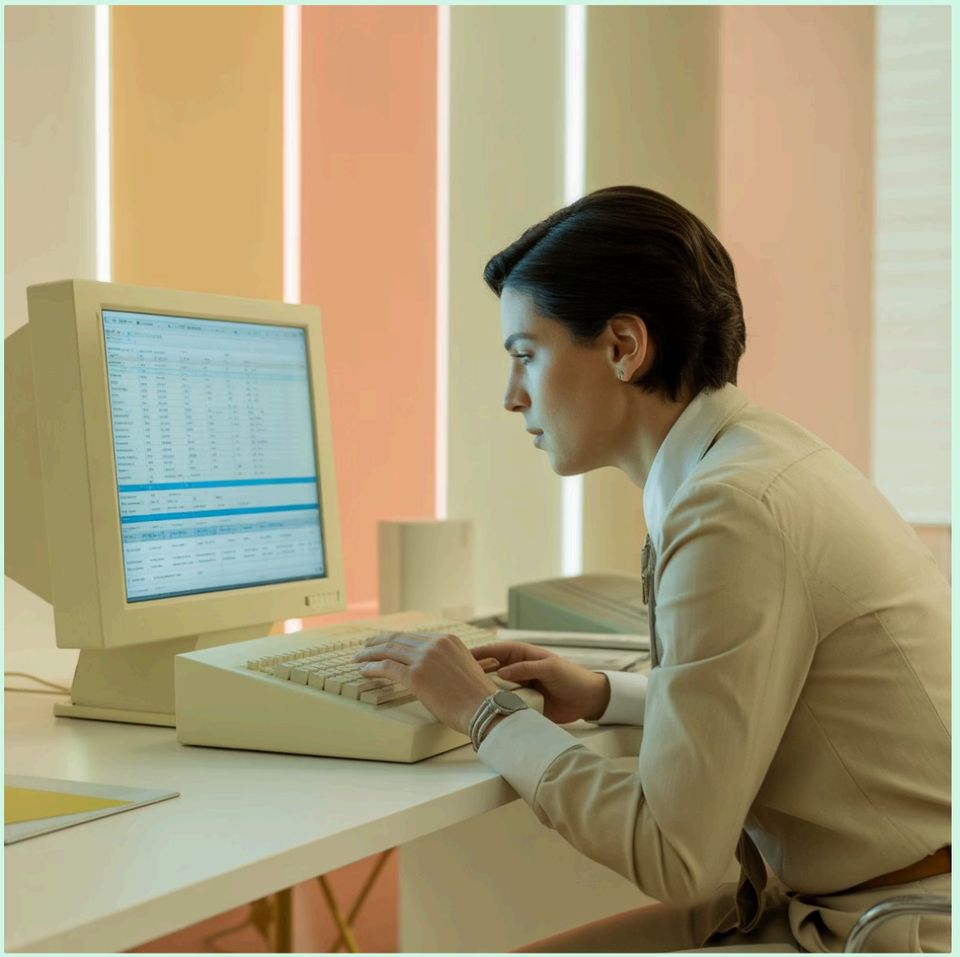
Lectura básica con csv.reader()

```
import csv

with open("alumnos.csv", "r") as archivo:
    lector = csv.reader(archivo)
    for fila in lector:
        print(fila)
```

Cada fila se devuelve como una lista. Por ejemplo: `['Ana', '9']`

La primera fila normalmente contiene los encabezados, que puedes omitir usando `next(lector)` si lo necesitas.



> 💡 **Tip:** Si solo necesitas los datos sin el encabezado, puedes usar `next(lector)` antes del bucle para saltarte la primera línea.

Lectura Avanzada: CSV como Diccionarios

Una de las características más poderosas del módulo `csv` es la capacidad de leer cada fila como un diccionario, donde las claves son los nombres de las columnas. Esto hace que el código sea mucho más legible y menos propenso a errores.

```
import csv

with open("alumnos.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        nombre = fila['nombre']
        nota = fila['nota']
        print(f"Nombre: {nombre}, Nota: {nota}")
```

Con `DictReader`, cada `fila` es un diccionario donde puedes acceder a los valores por nombre de columna en lugar de por índice numérico. Esto hace que tu código sea más claro y mantenible.

Más legible

Acceso por nombre de columna: `fila['nombre']`

Menos errores

No dependes de la posición exacta de las columnas

Más flexible

Cambios en el orden no rompen el código

TXT vs CSV: ¿Cuál elegir?

📄 Usa TXT cuando...

- Necesites guardar texto simple sin estructura fija
- Estés creando logs o registros de eventos
- Los datos no sigan un formato tabular
- Busques la máxima simplicidad

📊 Usa CSV cuando...

- Trabajes con datos tabulares (filas y columnas)
- Necesites compatibilidad con Excel u otras herramientas
- Vayas a hacer análisis de datos
- Tengas múltiples campos por registro

Ejemplo Integrador: Sistema de Gestión

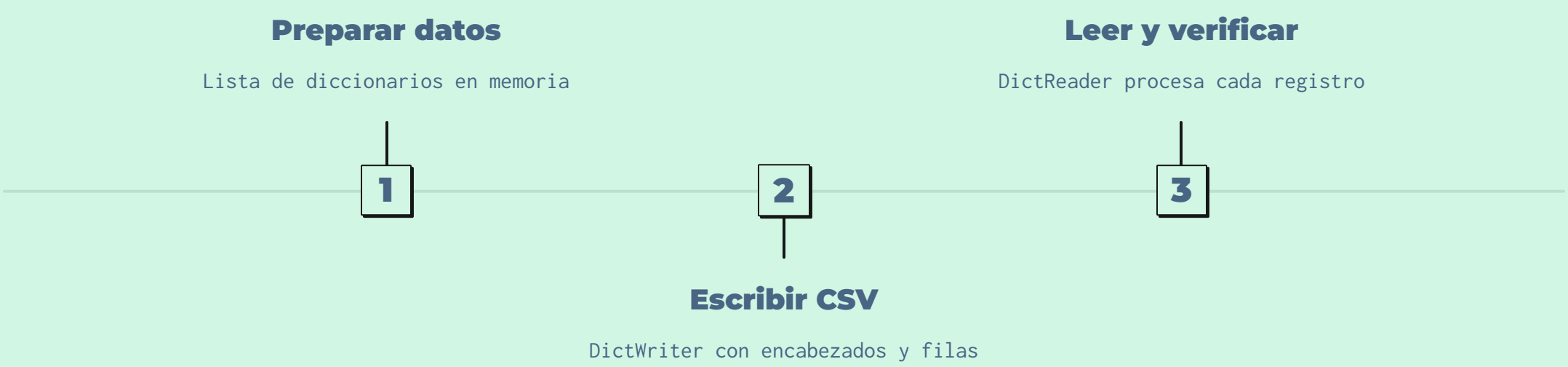
Veamos un ejemplo completo que combina lectura y escritura de archivos CSV usando diccionarios. Este patrón es muy común en aplicaciones reales.

```
import csv

# 1. Lista de diccionarios con los datos
alumnos = [
    {"nombre": "Ana", "nota": 9},
    {"nombre": "Luis", "nota": 8},
    {"nombre": "Carla", "nota": 10},
]

# 2. Escribimos el archivo CSV
with open("alumnos.csv", "w", newline="") as archivo:
    campos = ["nombre", "nota"]
    escritor = csv.DictWriter(archivo, fieldnames=campos)
    escritor.writeheader()  # Escribe los encabezados
    escritor.writerows(alumnos)  # Escribe todas las filas

# 3. Leemos el archivo para verificar
with open("alumnos.csv", "r") as archivo:
    lector = csv.DictReader(archivo)
    for fila in lector:
        print(f"{fila['nombre']} obtuvo {fila['nota']}")
```



🎯 Desafío Práctico: Gestión de Productos

Requisitos del programa

1 Guardar productos

Cada producto debe tener nombre y precio en productos.csv

2 Mostrar inventario

Al iniciar, mostrar todos los productos guardados

3 Agregar sin borrar

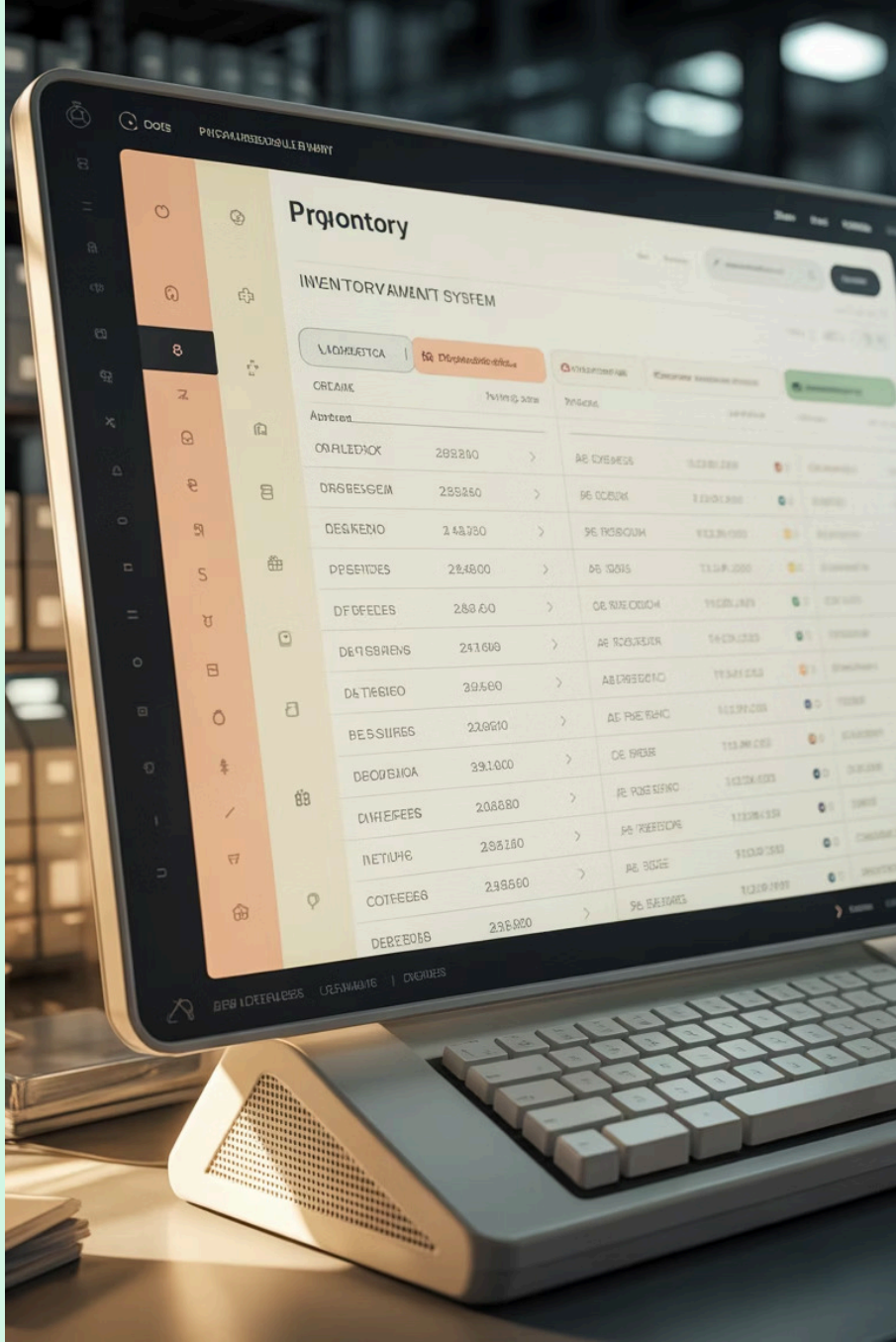
Permitir añadir más productos conservando los anteriores

💡 Desafío extra

- [ ] Calcula y muestra el **precio total** de todos los productos almacenados en el archivo. Pista: necesitarás convertir los precios de texto a números usando `float()` al leerlos.

Conceptos a aplicar:

- Lectura con DictReader
- Escritura con DictWriter en modo 'a'
- Conversión de tipos de datos
- Cálculos sobre datos leídos



Resumen y Mejores Prácticas

Usa siempre with

Garantiza el cierre automático de archivos y evita fugas de recursos

Maneja errores

Usa try-except para capturar FileNotFoundError y otros problemas

Elige el formato adecuado

TXT para simplicidad, CSV para datos estructurados

Aprovecha DictReader

Hace tu código más legible y mantenible al trabajar con CSV

La persistencia de datos es fundamental en cualquier aplicación real. Con estos conocimientos sobre archivos TXT y CSV, ya tienes las herramientas básicas para crear programas que guardan y recuperan información de forma efectiva. ¡A practicar!