Proyecto final: la herramienta ventas

En este proyecto construyes `ventas`, una herramienta de línea de comandos que lee un CSV de pedidos de PyShop, descarta las filas dañadas sin detenerse y muestra un reporte de ventas. Debe ser un paquete instalable: tras `pip install -e .` o `uv tool install .`, el comando `ventas pedidos.csv` funciona desde cualquier carpeta. El programa usa solo la biblioteca estándar de Python 3.11+; las pruebas pueden usar `pytest`. El archivo `ventas-cli.zip` contiene una solución de referencia completa: consúltalo al terminar, no antes.

Datos de entrada

Columnas: `id,fecha,cliente,ciudad,producto,cantidad`. Genera los datos con esta función, que siempre produce los mismos 1000 pedidos, con un 2 % de filas dañadas a propósito:

```
import csv
import io
import random
from datetime import date, timedelta

PRODUCTOS = {"Taza": 149.0, "Teclado": 899.0,
             "Ebook de Python": 299.0, "Mouse": 349.5,
             "Libreta": 89.0}
CIUDADES = ["CDMX", "Bogotá", "Lima", "Santiago", "Monterrey"]
PESOS = [30, 8, 20, 12, 30]


def generar_csv(n=1000, semilla=7):
    rng = random.Random(semilla)
    salida = io.StringIO()
    w = csv.writer(salida, lineterminator="\n")
    w.writerow(["id", "fecha", "cliente", "ciudad",
                "producto", "cantidad"])
    inicio = date(2026, 9, 1)
    for i in range(1, n + 1):
        fecha = inicio + timedelta(days=rng.randrange(30))
        cliente = f"C{rng.randrange(1, 201):03d}"
        ciudad = rng.choice(CIUDADES)
```

```
        producto = rng.choices(list(PRODUCTOS),
                               weights=PESOS)[0]
        cantidad = str(rng.randint(1, 5))
        r = rng.random()
        if r < 0.008:
            cantidad = "dos"
        elif r < 0.014:
            producto = "Gorra"
        elif r < 0.02:
            cantidad = "-1"
        w.writerow([i, fecha.isoformat(), cliente, ciudad,
                    producto, cantidad])
    return salida.getvalue()
```

Cambiar el orden de las llamadas a `rng` produce otras cifras.

Requisitos mínimos

1. **Estructura de paquete.** Un `pyproject.toml` con `requires-python = ">=3.11"` y la entrada `[project.scripts]` que crea el comando `ventas`. Código en `src/ventas/`, al menos en `modelos.py`, `lectura.py`, `reporte.py` y `cli.py`.
2. **Modelo inmutable.** `Pedido` es una `@dataclass(frozen=True, slots=True)` con `importe` como `@property`. Se ordena por importe con `__lt__` y `@total_ordering`.
3. **Validación con excepción propia.** Un `@classmethod desde_fila(fila, num_linea)` que lanza `FilaInvalidaError`, subclase de `ValueError` con los atributos `num_linea` y `motivo`: cantidad no numérica (con `raise ... from None`), cantidad menor o igual a cero o producto fuera del catálogo.
4. **Lectura tolerante.** Un generador `leer_pedidos(texto, errores)` que produce pedidos válidos y acumula los errores en la lista sin detenerse.
5. **Reporte.** Errores por motivo, venta total, piezas y venta por producto, venta por ciudad (de mayor a menor), mejor cliente y pedido más caro.
6. **Dos formatos.** Texto y JSON como clases intercambiables con el mismo método, sin cadenas de `if`.
7. **Interfaz con argparse.** `ventas ARCHIVO.csv [--top N] [--formato texto|json] [--salida RUTA]` y una opción para escribir el CSV de ejemplo, por ejemplo `ventas --demo pedidos.csv`. `main(argv=None)` recibe la lista de argumentos y devuelve el código de salida.
8. **Errores y avisos.** Archivo inexistente: mensaje en la salida de error y código 1, sin traza. Las filas descartadas se informan con `logging` (WARNING).

Cifras esperadas sobre generar_csv(1000, 7)

Tu herramienta debe reproducir exactamente estos valores:

- 1001 líneas con el encabezado; primera fila: `1,2026-09-11,C039,Santiago,Mouse,1`.
- 981 pedidos válidos y 19 errores: 9 por cantidad no numérica, 6 por cantidad no positiva y 4 por producto desconocido. El primero es `línea 111: cantidad no positiva -1`.
- Venta total: **718890.0**.
- Piezas: Taza 875, Libreta 836, Ebook de Python 564, Mouse 366 y Teclado 242.
- Venta por producto: Teclado 217558.0, Ebook de Python 168636.0, Taza 130375.0, Mouse 127917.0 y Libreta 74404.0.
- Venta por ciudad: CDMX 158538.0, Lima 157843.5, Bogotá 149033.0, Monterrey 134706.0 y Santiago 118769.5.
- 200 clientes distintos; el mejor es **C187**, con 11966.0 en 9 pedidos.
- Pedido más caro: 4495.0 (5 teclados); entre 10 empates, `max(pedidos)` devuelve el último, el id 980.
- `ventas no-existe.csv` termina con código 1; `ventas` sin argumentos, con código 2.

Una prueba mínima con pytest:

```
from ventas.cli import main


def test_reporte_json(tmp_path):
    csv_path = tmp_path / "pedidos.csv"
    salida = tmp_path / "reporte.json"
    main(["--demo", str(csv_path)])
    codigo = main([str(csv_path), "--formato", "json",
                   "--salida", str(salida)])
    assert codigo == 0
    assert "718890.0" in salida.read_text(encoding="utf-8")
```

Extras opcionales

- Venta por día y mejor día (resultado esperado: 2026-09-06, 34 pedidos, 32936.0).
- Un exportador CSV o Markdown sin modificar `reporte.py`.
- Filtros por ciudad o por fechas.
- Dinero con `decimal.Decimal` en lugar de `float`.
- Medir con `cProfile` la lectura de 100 000 pedidos y documentar qué optimizaste.
- Revisar el código con `ruff` y completar los type hints.

Entregables

1. **El paquete** en un zip o un repositorio: `pyproject.toml`, `src/ventas/`, `tests/` y un `README.md` con instrucciones de instalación y uso.
2. **Las pruebas** con las cifras esperadas, ejecutables con `pytest -q`.
3. **Una captura de terminal** (texto copiado): instalación, `ventas --demo pedidos.csv`, reporte en texto, reporte JSON con `--top 3` y `pytest`.
4. **Una nota breve** (media página) que explique dos decisiones de diseño: por qué `Pedido` es inmutable y por qué el cálculo del reporte está separado de los exportadores.

Rúbrica (100 puntos)

- **Empaquetado (15 puntos):** `pyproject.toml`, estructura `src/` y comando instalable.
- **Modelo y validación (20 puntos):** dataclass inmutable, `importe` calculado, orden y `FilaInvalidaError`.
- **Lectura y reporte (20 puntos):** generador tolerante, `Counter`/`defaultdict` y cifras correctas.
- **Exportadores e interfaz (15 puntos):** formatos intercambiables, `argparse` y códigos de salida.
- **Manejo de errores (10 puntos):** sin trazas, avisos con `logging`, ningún `except:` desnudo.
- **Pruebas (10 puntos):** las cifras esperadas y los casos de error pasan con `pytest`.
- **Claridad (10 puntos):** nombres claros, docstrings, README y nota de diseño.

Los extras no suman puntos, pero consolidan lo aprendido.