

# TP 3 Lab: Full OOP Toolbox — Interfaces (ABC), Inheritance, Polymorphism, Properties, Dunder Methods

---

## Goal

Build a small data pipeline where you can **swap data loaders** without changing the rest of the code.

You will implement:

- an **abstract base class** `BaseLoader` (contract/interface),
  - two concrete loaders: `CSVLoader` and `JSONLoader` (**inheritance**),
  - a `SalesDataset` that **uses a loader** (**composition**) and supports Pythonic behavior with **dunder methods**,
  - a `Sale` **@dataclass** with a **@property**,
  - a `SalesAnalyzer` that computes KPIs (pure logic).
- 

## Constraints

- Use a **virtual environment** (`venv`).
  - Use **standard library only** for the analysis (no pandas).
  - Use `abc.ABC` + `@abstractmethod` for the loader contract.
  - Dirty data rule:
    - missing required columns → **stop** (raise an error)
    - invalid row → **skip with a warning**
  - Keep code organized in `src/`.
- 

## Required Project Structure

session3/

```
main.py
requirements.txt
README.md
src/
  errors.py
  models.py
  loaders.py
  dataset.py
  analyzer.py
data/
  sales.csv
  sales.json
  sales_dirty.csv
  sales_dirty.json
out/
```

---

## Data Files (copy/paste)

Create these four files:

### data/sales.csv

```
date,product,quantity,unit_price
2026-01-01,coffee,2,1.80
2026-01-01,tea,1,2.10
2026-01-02,coffee,1,1.80
2026-01-02,chocolate,3,2.50
2026-01-03,tea,2,2.10
2026-01-03,coffee,4,1.80
```

### data/sales.json

```
[
  {"date":"2026-01-01","product":"coffee","quantity":2,"unit_price":1.8},
  {"date":"2026-01-01","product":"tea","quantity":1,"unit_price":2.1},
  {"date":"2026-01-02","product":"coffee","quantity":1,"unit_price":1.8},
  {"date":"2026-01-02","product":"chocolate","quantity":3,"unit_price":2.5},
  {"date":"2026-01-03","product":"tea","quantity":2,"unit_price":2.1},
  {"date":"2026-01-03","product":"coffee","quantity":4,"unit_price":1.8}
]
```

### data/sales\_dirty.csv

```
date,product,quantity,unit_price
2026-01-01,coffee,2,1.80
2026-01-01,tea,,2.10
2026-01-02,coffee,1,1.80
2026-01-02,chocolate,3,2,50
2026-01-03, tea ,2,2.10
2026-01-03,coffee,four,1.80
```

### data/sales\_dirty.json

```
[
  {"date":"2026-01-01","product":"coffee","quantity":2,"unit_price":1.8},
  {"date":"2026-01-01","product":"tea","quantity":null,"unit_price":2.1},
  {"date":"2026-01-02","product":"coffee","quantity":1,"unit_price":1.8},
  {"date":"2026-01-02","product":"chocolate","quantity":3,"unit_price":"2,50"},
  {"date":"2026-01-03","product":" tea ","quantity":2,"unit_price":2.1},
  {"date":"2026-01-03","product":"coffee","quantity":"four","unit_price":1.8}
]
```

---

## Tasks

### Task A — Domain Model (@dataclass) + Property (20 min)

In `src/models.py`, create:

- `@dataclass(frozen=True)` class `Sale`
    - fields: `date: str`, `product: str`, `quantity: int`, `unit_price: float`
    - property: `revenue -> float` returning `quantity * unit_price`
- 

### Task B — Custom Errors (10 min)

In `src/errors.py`, define:

- `class DatasetError(Exception)`
- `class SchemaError(DatasetError)` (missing required columns / invalid schema)
- `class ParseRowError(DatasetError)` (row can't be parsed)

---

### Task C — Interface + Inheritance: Loaders (35 min)

In `src/loaders.py`:

1. Create an abstract base class:
  - `class BaseLoader(ABC)`
    - `@abstractmethod def load(self, path: str) -> list[Sale]: ...`
2. Implement:
  - `class CSVLoader(BaseLoader)`
  - `class JSONLoader(BaseLoader)`

#### Requirements for both loaders

- Validate required keys/columns: `date`, `product`, `quantity`, `unit_price`
  - if missing → raise `SchemaError`
- Normalize product string: `.strip()`
- Convert types:
  - `quantity` → `int`
  - `unit_price` → `float` (bonus: accept `"2,50"` by replacing comma)
- Invalid row → print warning and skip it

---

### Task D — Dataset (Composition) + Dunder Methods (25 min)

In `src/dataset.py`, implement:

- `class SalesDataset`
  - `__init__(self, path: str, loader: BaseLoader)`
  - `load(self) -> None` fills `self.sales: list[Sale]`
  - dunder methods:
    - `__repr__(self) -> str`
    - `__len__(self) -> int` so `len(dataset)` works
    - `__iter__(self)` so `for sale in dataset:` works
    - `__getitem__(self, idx: int) -> Sale` so `dataset[0]` works

---

### Task E — Analyzer (Pure Logic) + Export (20 min)

In `src/analyzer.py`, implement:

- `class SalesAnalyzer`
    - `__init__(self, sales: list[Sale])`
    - `total_revenue(self) -> float`
    - `revenue_by_product(self) -> dict[str, float]`
    - `top_product(self) -> tuple[str, float]`
    - `export_revenue_by_product(self, path: str) -> None` (CSV export)
- 

## Task F — Main Program (CLI) (15 min)

In `main.py`:

- `parse`:
  - `--input` (required)
  - `--format` with choices `csv` or `json` (required)
- instantiate the correct loader:
  - `csv` → `CSVLoader()`
  - `json` → `JSONLoader()`
- load data via `SalesDataset`
- run analysis via `SalesAnalyzer`
- `print`:
  - total revenue
  - top product
- export `out/revenue_by_product.csv`

Example runs:

```
python main.py --format csv --input data/sales.csv
python main.py --format json --input data/sales.json
python main.py --format csv --input data/sales_dirty.csv
python main.py --format json --input data/sales_dirty.json
```

---

## Deliverables

- Working `session3/` project with the required structure
- `README.md` filled (template below)
- Exported file: `out/revenue_by_product.csv`
- Evidence that polymorphism works: same pipeline runs with CSV and JSON

---

## README Template (copy/paste into README.md)

# Session 3 — Advanced OOP Pipeline (CSV/JSON)

## Setup

```
```bash
```

```
python -m venv venv
```

```
# activate venv (OS-specific)
```

```
pip install -r requirements.txt
```

## Run

```
python main.py --format csv --input data/sales.csv
```

```
python main.py --format json --input data/sales.json
```

```
python main.py --format csv --input data/sales_dirty.csv
```

```
python main.py --format json --input data/sales_dirty.json
```

## Output

- Prints total revenue and top product
- Exports `out/revenue_by_product.csv`