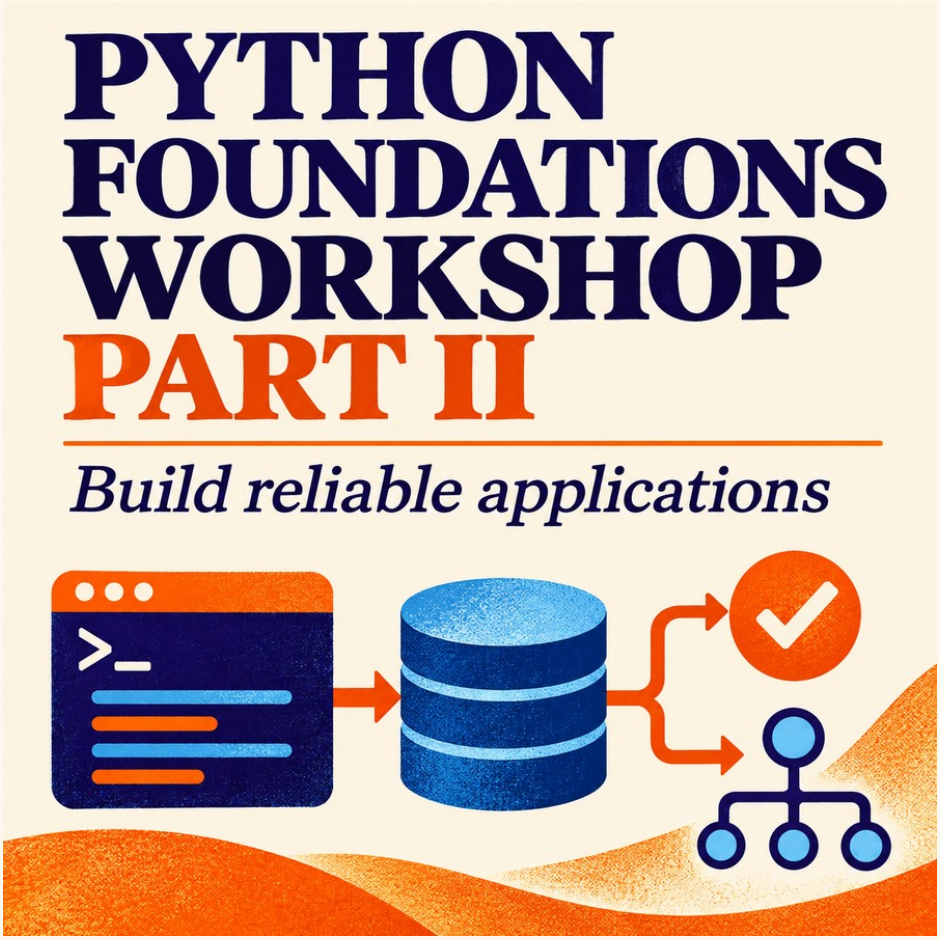
# Python Foundations Workshop II

**Build Reliable Applications**




96 runnable labs and three connected projects. Continue from basic Python into validated data, reliable storage, local services and maintainable releases.

**Companion website**

# Bring your Part I skills

You should be able to run a Python file, define a function, read a dictionary, handle an expected exception and recognize a class. Part I used an Expense Tracker to practice those skills. This volume adds a validated CSV feed, a replay-safe SQLite database, and a local HTTP summary.

### LOCAL SETUP

```
python --version
python -m venv .venv
```

Use Python 3.12 or newer for this edition. Use the virtual environment interpreter directly if your terminal does not activate it. On Windows it is .venv\Scripts\python.exe. On macOS and Linux it is .venv/bin/python. The bundled examples and projects use the standard library.

### HOW THE LABS WORK

Build from the task before turning to the reference. Check the recorded output and compare the executed lines. Add the separate assertion, reproduce the defect in a scratch copy, then implement the change request. Keep your smallest passing and failing cases.

### ONE SKILL TO REFRESH BEFORE CHAPTER 1

# Run the projects locally

Extract the complete ZIP, enter its projects folder, and run the following commands with the same interpreter. Use synthetic data and a fresh database. The HTTP service binds to loopback for practice.

```
python ledger.py inspect example.csv
python ledger.py import example.csv practice.db
python ledger.py summary practice.db
python -m unittest -v
python ledger.py serve practice.db --port 8000
```

The inspect command validates without creating a database. Import checks the entire feed before writing. Open the summary URL printed by serve, then stop the listener with Ctrl+C.

**Python language tutorial**

**SQLite reference**

**Concurrency with futures**

**Local HTTP server reference**

**Packaging metadata**

# Designing data contracts

Separate raw input, validated state and presentation. Use exact types, stable identity, units and explicit behavior for missing values.

**EIGHT LABS**

**01.01 Validate an imported record / 6**

**01.02 Parse a calendar contract / 15**

**01.03 Make duplicate identity explicit / 24**

**01.04 Keep missing and null separate / 33**

**01.05 Preserve display and lookup forms / 42**

**01.06 Choose a stable sort contract / 51**

**01.07 Compare exact decimal rates / 60**

**01.08 Freeze a domain value / 69**

Choose the layer that rejects bad data. A forgiving input adapter can normalize harmless differences while the domain layer preserves hard invariants.

# Validate an imported record

## Build brief

### ASSIGNMENT

An importer accepts exactly category and cents. Preserve a cleaned category and reject a Boolean amount.

### CHAPTER CONTEXT

Separate raw input, validated state and presentation. Use exact types, stable identity, units and explicit behavior for missing values.

### INPUTS, RETURN SHAPE, AND FIRST EXPECTED RESULT

# Validate an imported record

## Reference implementation

```
def clean(row):
    if set(row) != {"category", "cents"}:
        raise ValueError("unexpected fields")
    name, cents = row["category"], row["cents"]
    if not isinstance(name, str) or not name.strip():
        raise ValueError("category required")
    if type(cents) is not int or cents <= 0:
        raise ValueError("positive integer cents required")
    return name.strip().casefold(), cents
print(clean({"category": " Rail ", "cents": 475}))
```

Run the bundled file with Python before trying a variation.

`labs/ch01_01.py`

# Validate an imported record

## Observed behavior and trace

### RECORDED OUTPUT

```
('rail', 475)
```

### EXECUTED LINE NUMBERS AND COUNTS

```
01 x 1    02 x 1    04 x 1    05 x 1    07 x 1    09 x 1
10 x 1
```

Trace counts follow the current Python thread. Some callbacks or worker bodies can run in another thread. Compare the output with the contract even if some worker lines are absent from this trace.

### FIND THE LINE RESPONSIBLE FOR A VISIBLE OUTPUT

# Validate an imported record

## Verification

Run this check after the reference implementation. The bundled script includes the check, and a successful run means the assertion passed as well as producing the recorded output.

```
assert clean({"category":" BOOK ","cents":40}) == ("book",40)
```

### DESIGN A BOUNDARY CHECK

An isinstance check for int accepts True because bool subclasses int.

### SMALLEST INPUT, EXPECTED RESULT, AND WHAT THE ASSERTION PROVES

# Validate an imported record

## Diagnose a failure

### DEFECT TO INVESTIGATE

An isinstance check for int accepts True because bool subclasses int.

### REPRODUCE IN A SCRATCH COPY

Write a counterexample based on the defect. Run it first against the reference, then make one deliberately flawed edit that introduces the defect. Record the changed output or exception before restoring the reference.

### ORIGINAL LINE AND CHANGED LINE

### OBSERVED DIFFERENCE

# Validate an imported record

**Compare design choices**

### ENGINEERING CHOICE

Choose the layer that rejects bad data. A forgiving input adapter can normalize harmless differences while the domain layer preserves hard invariants.

### APPLY IT HERE

An importer accepts exactly category and cents. Preserve a cleaned category and reject a Boolean amount.

Identify one decision made in the implementation. Write an alternative and a fixture that would help a reviewer choose between them. Preserve the base behavior while comparing.

### ALTERNATIVE AND DECISION CRITERION

# Validate an imported record

## Implement the change request

### NEW REQUIREMENT

Add a caller-facing error that identifies a row number without copying private text.

### PRESERVE A PASSING BEHAVIOR

```
assert clean({"category":" BOOK ","cents":40}) == ("book",40)
```

Decide whether this original check still describes the supported interface. Add a new check for the changed behavior and keep it alongside the original when both contracts apply.

### CHANGE AND NEW VERIFICATION

# Validate an imported record

**Integrate it with a caller**

### CALLER CONTRACT

An importer accepts exactly category and cents. Preserve a cleaned category and reject a Boolean amount.

Sketch a second caller that supplies data from a terminal command, an import file, or a fixture. Keep the calculation independent from that input boundary. State which errors the caller can handle and which should stop the workflow.

### REVIEW THE HANDOFF

Add a caller-facing error that identifies a row number without copying private text.

### INPUTS, OUTPUT, AND A FAILURE THE CALLER MUST HANDLE

# Validate an imported record

**Acceptance record**

Submit the original run, one counterexample, and one version that handles the change request. Keep the script and invocation together so another reader can reproduce your result.

**EXPECTED BEHAVIOR**

An importer accepts exactly category and cents. Preserve a cleaned category and reject a Boolean amount.

**REGRESSION TO CATCH**

An isinstance check for int accepts True because bool subclasses int.

**COMMAND, OBSERVED RESULT, AND REMAINING LIMIT**

# Parse a calendar contract

## Build brief

### ASSIGNMENT

Accept an exact ISO calendar day, then compare it with a fixed cutoff.

### CHAPTER CONTEXT

Separate raw input, validated state and presentation. Use exact types, stable identity, units and explicit behavior for missing values.

### INPUTS, RETURN SHAPE, AND FIRST EXPECTED RESULT

# Parse a calendar contract

## Reference implementation

```
from datetime import date
def on_or_after(raw, cutoff):
    day = date.fromisoformat(raw)
    if day.isoformat() != raw:
        raise ValueError("use YYYY-MM-DD")
    return day >= cutoff
print(on_or_after("2027-01-01", date(2026, 12, 31)))
```

Run the bundled file with Python before trying a variation.

`labs/ch01_02.py`

# Parse a calendar contract

## Observed behavior and trace

### RECORDED OUTPUT

```
True
```

### EXECUTED LINE NUMBERS AND COUNTS

```
01 x 1    02 x 1    03 x 1    04 x 1    06 x 1    07 x 1
```

Trace counts follow the current Python thread. Some callbacks or worker bodies can run in another thread. Compare the output with the contract even if some worker lines are absent from this trace.

### FIND THE LINE RESPONSIBLE FOR A VISIBLE OUTPUT