# Python Foundations Workshop V

**Turn Data Into Decisions**




96 runnable cases across 48 operating patterns, with three connected projects on validated data, defensible metrics and reproducible reporting.

**Companion website**

# Finish the series with a data project

Use Part I's Python and data structures, Part II's contracts and tests, Part III's reproducibility, and Part IV's browser presentation. This final volume turns synthetic Workshop Desk booking records into an attendance report that a reviewer can rebuild and audit.

## 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

Work through pairs of cases. They share an implementation and test different conditions. Predict the output and compare the trace. Run the assertion, reproduce a failure in a scratch copy, then make the change request.

## 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 the synthetic input folder and a new destination directory for each run.

```
python -m unittest -v
python report_cli.py sample report-first
python report_cli.py sample report-second
# Open report-first/report.html in a browser.
```

The first two builds should produce byte-identical report and manifest files. Review missing dates, denominator rules, and suppressed values before sharing an aggregate report.

**Python statistics reference**

**Python zoneinfo reference**

**NIST statistical methods handbook**

**W3C accessible complex images**

# Data grain, keys and measurement units

Identify the meaning of one row before counting. Check keys, units and missing values. Keep an event log distinct from a snapshot.

FOUR PATTERNS AND EIGHT CASES

**01.01 Count rows and distinct bookings / 6**

**01.02 Check a one-row-per-booking export / 15**

**01.03 Verify a composite attendance key / 24**

**01.04 Detect a duplicate fact key / 33**

**01.05 Reconcile decimal dollars and cents / 42**

**01.06 Refuse a mismatched unit conversion / 51**

**01.07 Separate zeros from unknown values / 60**

**01.08 Preserve an entirely missing measure / 69**

A row count is cheap to compute. A distinct count is closer to the business question only when the identity contract is sound.

# Count rows and distinct bookings

## Build brief

### ASSIGNMENT

Count rows and distinct bookings. Predict the result and identify the input boundary that controls this case.

```
decide([("B1","booked"),("B1","checked_in"),("B2","booked")])
```

### CHAPTER CONTEXT

Identify the meaning of one row before counting. Check keys, units and missing values. Keep an event log distinct from a snapshot.

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

# Count rows and distinct bookings

## Reference implementation

```
def decide(rows):
    return len(rows), len({row[0] for row in rows})
result = decide([("B1","booked"),("B1","checked_in"),("B2","booked")])
print(repr(result))
```

Run the bundled file with Python before trying a variation.

`labs/ch01_01.py`

# Count rows and distinct bookings

## Observed behavior and trace

### RECORDED OUTPUT

```
(3, 2)
```

### EXECUTED LINE NUMBERS AND COUNTS

```
01 x 1    02 x 4    03 x 1    04 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

# Count rows and distinct bookings

### 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.

```
expected = (3,2)
assert type(result) is type(expected)
assert result == expected
```

### DESIGN A BOUNDARY CHECK

A count of event rows is not automatically a count of reservations.

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

# Count rows and distinct bookings

## Diagnose a failure

### DEFECT TO INVESTIGATE

A count of event rows is not automatically a count of reservations.

### 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

# Count rows and distinct bookings

## Compare design choices

### ENGINEERING CHOICE

A row count is cheap to compute. A distinct count is closer to the business question only when the identity contract is sound.

### PAIRED CASE

Check a one-row-per-booking export

```
decide([("B1","booked"),("B2","booked")])
# Expected: (2,2)
```

Compare the paired case with the original. Name the condition that changed and explain which behavior should remain invariant.

### ALTERNATIVE AND DECISION CRITERION

# Count rows and distinct bookings

## Implement the change request

### NEW REQUIREMENT

Define a separate event key and show how to audit repeated booking IDs without treating them as defects.

### PRESERVE A PASSING BEHAVIOR

```
expected = (3,2)
assert type(result) is type(expected)
assert result == expected
```

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

# Count rows and distinct bookings

**Integrate it with a caller**

### CALLER CONTRACT

Count rows and distinct bookings. Predict the result and identify the input boundary that controls this case.

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

Define a separate event key and show how to audit repeated booking IDs without treating them as defects.

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

# Count rows and distinct bookings

**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**

Count rows and distinct bookings. Predict the result and identify the input boundary that controls this case.

**REGRESSION TO CATCH**

A count of event rows is not automatically a count of reservations.

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

# Check a one-row-per-booking export

## Build brief

### ASSIGNMENT

Check a one-row-per-booking export. Predict the result and identify the input boundary that controls this case.

```
decide([("B1","booked"),("B2","booked")])
```

### CHAPTER CONTEXT

Identify the meaning of one row before counting. Check keys, units and missing values. Keep an event log distinct from a snapshot.

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

# Check a one-row-per-booking export

## Reference implementation

```
def decide(rows):
    return len(rows), len({row[0] for row in rows})
result = decide([("B1","booked"),("B2","booked")])
print(repr(result))
```

Run the bundled file with Python before trying a variation.

`labs/ch01_02.py`

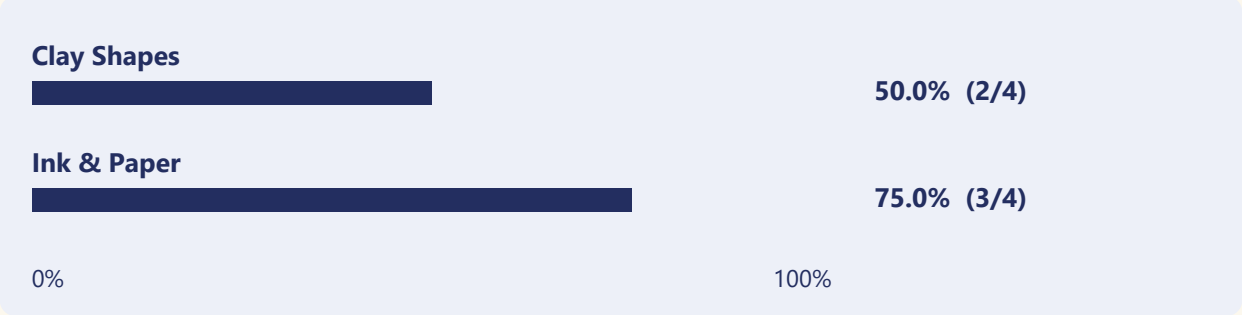
# Draw a chart from the same values

## Run the checkpoint

### VERIFICATION CASE

Compare the Ink 75.0 percent bar with the Clay 50.0 percent bar. Confirm their table cells use the same values.

### WORKED RESULT FROM THE SAMPLE DATASET




The printed counts show the same numerators and denominators as the HTML data table.

Record the interpreter, command, input fixture and result. Use a fresh output directory if the checkpoint writes report files. Compare the observed output and exit status with the contract.

### EXPECTED AND ACTUAL STATUS

### WHAT THIS CHECKPOINT LEAVES UNTESTED