Lecture 13 · Practice Exercises

Error Handling and Debugging

Exercise structure and progressive support

Guided (1): every logical stage is explained and scaffolded in the starter.

Laboratory (2): implement essential functions with targeted hints in the starter.

Individual (3): exam-style specifications with minimal implementation support.

All exercises specify context, input contract, required behaviour, output and edge cases.

Reference data are tests, never values to hard-code.

The problem statement must be sufficient even when the instructor is not present.

Guided · Debugging a Transaction Processor

Exercise 1 / 6 · Lecture 13

Guided: problem context and inputs

PROBLEM CONTEXT

A transaction import routine receives amounts as strings. An invalid record can crash the original script; an incorrect accumulator can corrupt the total.

INPUT CONTRACT

List of strings, e.g. ["10", "bad", "-3"]. Numeric strings may be positive or negative; nonnumeric strings should be skipped and counted.

REQUIRED BEHAVIOUR

Inspect the deliberately buggy code, identify the defects, implement narrow `ValueError` handling, and return both total and skipped count.

OUTPUT / REFERENCE TEST

```
Reference → total 7.00, skipped 1. All invalid →
total 0.00, skipped equals input count.
```

A sample input illustrates the format; your solution must handle every input satisfying this contract.

Guided: implementation walkthrough

FOLLOW THE STAGES

1. Identify the input source and expected types
2. Choose functions with one responsibility each
3. Implement the provided reference case
4. Inspect intermediate results and the output
5. Change the input and run a new check

Guided: reproduce traceback, name the failing operation, fix accumulator, add one targeted handler.

WHY / EDGE CASE

DESIGN REASON

Why `ValueError`? `float()` rejects malformed numeric strings.

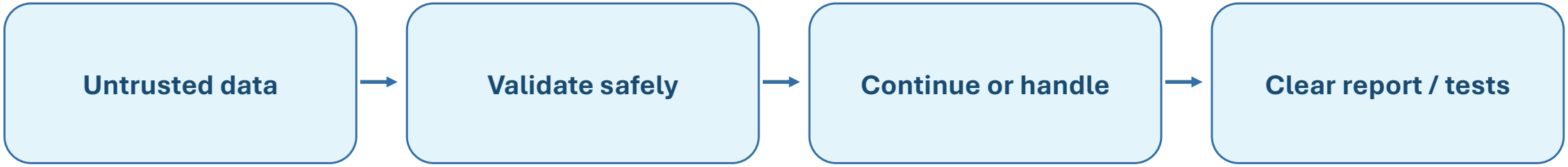
EDGE CASE TO TEST

Test [], ["10"], and ["bad"].

The Guided starter contains an explicit scaffold: follow and understand it step by step.

Debugging a Transaction Processor: input → processing → output

Follow the data through the program before writing code.



Write the contracts before coding; one example is not the full specification.

Laboratory 1 · Robust CSV Reader

Exercise 2 / 6 · Lecture 13

Laboratory 1: problem context and inputs

PROBLEM CONTEXT

Daily sales files may be absent or contain amounts that are not valid numbers. The program must explain expected problems without a raw traceback.

INPUT CONTRACT

Input sales CSV with header amount, e.g. 12; NA; 8. A source path is given to load_sales(path).

REQUIRED BEHAVIOUR

Read with DictReader, catch FileNotFoundError for missing files, skip rows with bad amount fields, count valid and rejected rows, and report the total.

OUTPUT / REFERENCE TEST

`File with 12,NA,8 → valid count 2, rejected 1, total 20.00. Missing path → clear message, no crash.`

Laboratory: targeted hints and tests

HINTS

1. Hint: catch file opening failure separately from row conversion failure.
2. Decide which part belongs in a separate function.
3. Test normal and boundary data.
4. Verify generated messages and files.

WHY / EDGE CASE

DESIGN REASON

Avoid a single broad except around the whole function.

EDGE CASE TO TEST

Test wrong CSV header and empty file.

Laboratory 2 · Financial Calculator Debugging

Exercise 3 / 6 · Lecture 13

Laboratory 2: problem context and inputs

PROBLEM CONTEXT

A colleague wrote a simple-interest calculator. It runs but the interest is wrong for a small known case. You need to fix the logic and reject invalid assumptions.

INPUT CONTRACT

Function inputs: principal (nonnegative), annual rate (nonnegative fraction), years (nonnegative). Simple interest is principal×rate×years.

REQUIRED BEHAVIOUR

Run the buggy function, reproduce the discrepancy, locate the wrong operation with debugger/print, fix it, and add domain validation.

OUTPUT / REFERENCE TEST

`P=1000, rate=.05, years=2 → interest 100.00 and final 1100.00. Try a zero-year loan.`

Laboratory: targeted hints and tests

HINTS

1. Hint: compare intermediate values with the hand calculation.
2. Decide which part belongs in a separate function.
3. Test normal and boundary data.
4. Verify generated messages and files.

WHY / EDGE CASE

DESIGN REASON

This is a logic error: the program runs but violates the formula.

EDGE CASE TO TEST

Test negative rate and years.

Individual 1 · Safe Data Converter

Exercise 4 / 6 · Lecture 13

Individual 1: problem context and inputs

PROBLEM CONTEXT

A data-import process receives numeric values as text and cannot reject the entire batch just because some entries are malformed.

INPUT CONTRACT

Input is a list of strings, e.g. ["10.5", "", "x", "-2"]. Whitespace is allowed; invalid entries do not contribute to the result.

REQUIRED BEHAVIOUR

Implement parse_numbers(texts) returning (valid floats list, invalid count). Handle only expected numeric-conversion failures.

OUTPUT / REFERENCE TEST

```
Reference → [10.5, -2.0], invalid count 2. Empty
input → [], 0.
```

Independent: verification and corner cases

SELF-CHECK

Follow the specified input contract.

Hint: isolate the risky float conversion.

Compare output with the reference test.

Verify that your code works with other valid inputs.

WHY / EDGE CASE

DESIGN REASON

Use a result list and a separate counter.

EDGE CASE TO TEST

Test all-invalid and whitespace-only entries.

Individual 2 · Faulty Portfolio Analyzer

Exercise 5 / 6 · Lecture 13

Individual 2: problem context and inputs

PROBLEM CONTEXT

A teammate's analysis function contains two deliberate logic errors and lacks an empty-input validation rule. Your task is to diagnose rather than rewrite blindly.

INPUT CONTRACT

Input list of numeric asset contributions, e.g. [2, -1, 3]. Negative values are allowed; empty list is invalid.

REQUIRED BEHAVIOUR

Inspect and fix the provided function; return dictionary with total and mean; raise ValueError for empty input. Describe how reference tests exposed the defects.

OUTPUT / REFERENCE TEST

`[2,-1,3] → total 4 and mean 1.333... ; [5] → total 5, mean 5; [] raises ValueError.`

Independent: verification and corner cases

SELF-CHECK

Follow the specified input contract.

Hint: write expected values on paper before stepping through the loop.

Compare output with the reference test.

Verify that your code works with other valid inputs.

WHY / EDGE CASE

DESIGN REASON

Only change the lines that violate requirements.

EDGE CASE TO TEST

Test a single-element list and negatives.

Individual 3 · Reliable Report Generator

Exercise 6 / 6 · Lecture 13

Individual 3: problem context and inputs

PROBLEM CONTEXT

A reporting application reads a CSV with occasional malformed values. It should still create a summary from valid data and give useful feedback when the input file is missing.

INPUT CONTRACT

source.csv contains header amount, e.g. 12; invalid; -2. Input and destination file paths are supplied.

REQUIRED BEHAVIOUR

Read CSV, catch FileNotFoundError, skip malformed rows, compute valid count, rejected count and sum, and write a persistent text report.

OUTPUT / REFERENCE TEST

```
Reference: valid count 2; rejected count 1; total
10.00. Missing file: no report written and clear
feedback.
```

Independent: verification and corner cases

SELF-CHECK

Follow the specified input contract.

Hint: keep file opening, row parsing and report writing conceptually separate.

Compare output with the reference test.

Verify that your code works with other valid inputs.

WHY / EDGE CASE

DESIGN REASON

Do not catch every exception indiscriminately.

EDGE CASE TO TEST

Test valid-only CSV, header-only CSV and missing file.

Independent starters are intentionally minimal: you choose the design.

Final verification checklist

- The program handles inputs beyond the supplied example.
- The required files and output names match the specification.
- Functions have clear responsibilities; reusable code is separate from interface operations.
- At least one boundary or invalid case is tested.
- The starter is your workspace; the instructor solution is a later comparison.

A fully specified problem can still admit many correct implementations.