Lab 12 · Part 5 Robustness and the Real World · AI Level 1

Processing Real Text Data

Build a reproducible log pipeline with field validation, actionable diagnostics, statistics and CSV round-trip checks.

Due: Homework 4 · Tue 8 Dec, 23:59

Objectives

- Reproduce a random experiment under recorded conditions
- Read and write files with an explicit encoding and with
- Parse named fields and distinguish syntax from semantic validation
- Account for every input line and verify exported records

Allow about 40 minutes for A, 35 for B and 15 for C. All three are available together. Submit code and answer cards in Homework 4. AI policy is Level 1: declare actual assistance or nonuse, revisions and executed verification. Do not submit passwords, private keys or session tokens.

Data and Contract

Download the log dataset and save it as `lab12/app.log`. The UTF-8 file contains 212 physical lines: 200 valid records and 12 rejected lines. A final newline does not add an extra line. Blank lines and comments count as rejected lines. The generator uses Python 3.13 and seed 1602; replay also requires the same sequence of calls.

```
2026-11-05 14:23:45 INFO user=alice action=login duration=1234ms
```

This lab defines a teaching subset:

- Date is ASCII `YYYY-MM-DD` and a valid calendar date. Time is ASCII `HH:MM:SS` with valid hour, minute and second.

- Level is `INFO/WARN/ERROR`; user follows `[A-Za-z][A-Za-z0-9_]*`; action contains one or more ASCII lowercase letters.
- Fields are separated by one or more spaces or Tabs. No embedded newline, leading/trailing whitespace, or trailing garbage is accepted.
- Duration is a nonnegative ASCII integer followed by `ms`, with value 0–3000. Return only its digit string.
- Use named groups and `re.fullmatch` or equivalent whole-record matching. Validate dates and times with `datetime.date.fromisoformat` and `datetime.time.fromisoformat`. Catch expected `ValueError`, not arbitrary programming errors.

A · Encoding and Individual Records

12-1 Encoding Experiment

Implement `encoding_lengths(text: str) -> tuple[int, int]`, returning UTF-8 and GBK byte lengths. Let `UnicodeEncodeError` propagate if GBK cannot represent a character; do not replace or ignore it.

On the A card, include executed code and output for three checks: the two lengths of `"计算导论"`; decoding the UTF-8 bytes of `"你好"` as Latin-1 and reversing that mapping; the complete exception from ASCII decoding. Explain why no exception does not establish correct text.

12-2 Reading a File Safely

```
def read_lines(path: str) -> list[str]:
    """Return UTF-8 lines without trailing newlines."""
```

Use `with` and direct iteration, not `readlines()`. Preserve other whitespace. Keep blank lines as empty strings and process a final line without a newline. A missing file raises `FileNotFoundError` containing the path, with the original cause retained.

On the card, explain why returning every line still uses growing memory, and how the processing could change if only a count were needed.

12-3 Parsing One Log Record

```
def parse_line(line: str) -> dict[str, str] | None:
    """Input has no trailing newline.
    Return six string fields, or None.
    """
```

The keys are `date/time/level/user/action/duration`. Follow the shared contract, use named groups, and validate everything before returning. Return `None` for blanks, comments, missing fields, invalid dates/times, non-ASCII digits, out-of-range durations and trailing garbage. Prepare at least three positive and five negative cases.

12-4 Extracting Contacts

Implement these functions, preserving occurrence order and duplicates:

```
def extract_emails(text: str) -> list[str]: ...
def extract_phones(text: str) -> list[str]: ...
def email_domains(text: str) -> list[str]: ...
```

This is a teaching subset, not the complete email standard. Candidates are separated by whitespace, Chinese/English commas, semicolons, colons or `/`. The email local part contains one or more `[A-Za-z0-9._-]` characters. Its domain has at least two nonempty `[A-Za-z0-9-]+` labels separated by dots. A phone is eleven ASCII digits beginning with 1. Match each entire candidate rather than extracting a substring of an invalid token. `email_domains` returns the portion after `@` in accepted emails.

```
Alex: alex@sjtu.edu.cn 13812345678
Blake: blake.wang@example.com, 15900001111
Repeat: alex@sjtu.edu.cn
Invalid: not-an-email@ / 123456 / x13812345678y
```

12-5 Reproducible Dice Rolls

```python
def roll_dice(n: int, seed: int) -> list[int]: ...
```

Create a local `random.Random(seed)` and call `randint(1, 6)` exactly n times in order. Return an empty list for zero and raise `ValueError` for negative n. Initialize once per function call, not before every draw, and preserve the module-level random state. Identical parameters replay under the same Python version.

B · One Complete Log Pipeline

12-6 Loading, Diagnostics and Statistics

Reuse your completed `parse_line` in the same submitted file and implement:

```python
def load_log(path: str) -> tuple[
    list[dict[str, str]],
    list[tuple[int, str]],
]: ...
def count_by_level(records: list[dict[str, str]]) -> dict[str, int]: ...
def busiest_user(records: list[dict[str, str]]) -> str: ...
def average_duration(records: list[dict[str, str]]) -> float: ...
```

Read UTF-8 line by line. Return accepted records and rejected `(1-based physical line number, original text without the newline)` pairs, both in order. Reject blanks and comments as well. Preserve every valid record. Propagate file access, permission and decoding errors instead of swallowing them as record failures. Report rejected inputs with path, line, raw text and reason; if only a format mismatch is known, show the expected structure rather than inventing a field-specific cause.

Statistics receive already validated records. Do not arbitrarily filter them again. `count_by_level` includes only observed levels. `busiest_user` selects the most frequent user, breaking ties by smallest lexicographic name, and returns `""` for empty input. `average_duration` uses all accepted records as its denominator and returns `0.0` for empty input. Do not modify inputs.

On the B card, report accepted/rejected counts, each rejected position and diagnostic, and the total. Show a complete path from one bad input to its diagnostic. Retaining all records is required by this interface and must not be described as constant-memory processing.

12-7 CSV Export and Read-Back

```
def write_csv(records: list[dict[str, str]], path: str) -> None: ...
```

Use `csv`, UTF-8 and `newline=""`. Write headers in the order `date,time,level,user,action,duration`, preserving record order. Empty input still writes a header. An existing output may be overwritten; verify that the path is an output before calling. Propagate file errors.

Read back with `csv.DictReader` and assert equality of complete records, not merely counts. Separately test fields containing commas or quotes to check CSV quoting; this does not relax the log parser's username contract.

B-1 Failure Boundaries and Tests

Submit at least five negative or boundary cases covering invalid dates, empty users, invalid/out-of-range durations, trailing garbage and a final line without a newline. Include one complete diagnostic and CSV round-trip output. Explain why a file-level failure must not become an unexplained empty record list.

12-8 Extension: Birthday Simulation

Retain `birthday_collision(n_people, trials, seed=None) -> float` as an extension. Assume independent uniform birthdays on 365 days; initialize once per experiment. Compare estimates for 20–30 people and save two runs with identical conditions. A finite simulation need not first cross 50% at 23, so that fluctuating threshold is not the sole correctness criterion. This extension does not add a required code submission.

C · SSH and scp

C-1 Remote File Processing

Use the practice host and your own practice account supplied by the course staff. Uppercase words below are placeholders. If accounts are not yet available, complete local script/path checks and state which remote steps remain, without inventing evidence.

```
ssh USER@HOST
mkdir -p cs1602-lab12
exit
scp lab12/app.log USER@HOST:cs1602-lab12/app.log
scp analyze.py USER@HOST:cs1602-lab12/analyze.py
ssh USER@HOST
cd cs1602-lab12
python3 analyze.py
exit
scp USER@HOST:cs1602-lab12/parsed.csv lab12/parsed-remote.csv
```

Make the script read `app.log` and write `parsed.csv` using relative paths. `scp` copies files; `ssh` opens a remote command session. Check the first-connection host fingerprint against course staff information. Keep passwords, private keys and tokens out of reports. The C card records redacted commands, the remote working directory, read-back verification and unfinished steps.

Solo Assignment 3

Big integers begins this week and spans weeks 12–13. Use string interfaces for addition, subtraction, multiplication, division and powers, with a technical report. Consult the submission page and schedule for the exact statement and configured deadline.

Before Submitting

- Complete five A code tasks and the A card; two B code tasks and the B card.
- Match all seven code interfaces, return shapes and exception contracts.

- The standard data gives 200 accepted and 12 rejected lines, totaling 212, with locatable diagnostics.
- Compare full CSV records and keep diagnostics out of data files.
- Record actual C progress and AI verification without account secrets.