**Project: “ReadRack” — A CLI Personal Library & Reading Tracker**

**1) Detailed problem statement**

You are building **ReadRack**, a command-line app to help a user manage a small personal library and track reading progress. Users can:

* Import books from a CSV file and persist their library to disk.
* Add, edit, and remove books.
* Search/filter by title/author/genre using string operations.
* List unique authors/genres and get quick stats (e.g., total books, completed vs. in-progress).
* Track reading progress per book (status, pages read, rating, notes).
* Iterate through books by different strategies (e.g., by genre, unread first) using a **custom iterator**.
* Save session changes back to files (CSV/JSON).
* Organize code into modules and use classes/objects cleanly.

**Constraints**

* Pure Python standard library only.
* Works on any OS with Python 3.10+.
* No DB; use files (CSV for import/export, JSON for state).
* Clear separation of concerns via modular design.

**2) Technical scope (what to include)**

**Core Python & Getting Started**

* Entry point script (main.py), help text, simple menu loop.

**Working with Strings**

* Case-insensitive search, substring and prefix/suffix matches.
* Sanitizing inputs (strip/replace), title-casing, splitting/joining tags.

**Iterator Protocol**

* A LibraryIterator that implements \_\_iter\_\_ and \_\_next\_\_.
* Multiple iteration modes (e.g., mode="genre" groups, mode="unread").

**Lists, Tuples, Sets**

* **Lists** for the book collection order.
* **Tuples** for immutable summaries (e.g., (total, completed, in\_progress)).
* **Sets** for unique authors/genres and to deduplicate imported entries.

**Dictionaries**

* Index maps: id -> Book, author -> [Book], genre -> [Book].
* Config/preferences map (e.g., default view mode).

**File Handling**

* **Read** initial data from /data/books.csv.
* **Write** library state to /data/library.json.
* **Export** filtered results back to CSV.
* Basic error handling for missing files/invalid rows.

**Functions**

* Small, single-purpose functions (parse row, validate input, compute stats).

**Modular Development**

* Separate modules (I/O, models, iterators, services, cli).
* Avoid circular imports; expose clean APIs.

**Classes and Objects**

* Book class with methods (mark\_read, add\_note, etc.).
* Library aggregate managing book list, indices, and persistence.

**3) High-level approach (suggested steps)**

1. **Scaffold the project**
   * Create folders/files (see structure below).
   * Stub modules with TODOs and minimal function/class skeletons.
2. **Model layer**
   * Implement Book dataclass-style class (id, title, author, genre, pages, status, rating, notes).
   * Implement Library with add/remove/update, search, compute stats, and persistence.
3. **Iterator**
   * Implement LibraryIterator to yield Book objects in the chosen strategy.
   * Add iteration mode selection on the CLI (e.g., list --mode unread).
4. **I/O & persistence**
   * CSV import (skip duplicates using a set of (title.lower(), author.lower())).
   * JSON save/load for state (pretty print).
5. **CLI**
   * Menu loop (text options).
   * Commands: import, list, search, add, update, remove, stats, export, quit.
   * Parse arguments, validate inputs, print friendly messages.
6. **Testing by usage**
   * Use small sample CSV.
   * Try edge cases: empty library, duplicate imports, invalid ratings/pages.
7. **Polish**
   * Help text and error messages.
   * Docstrings/readme, clear prompts.

**4) Proposed folder structure**

readrack/

README.md

main.py # entry point (menus/commands)

models/

\_\_init\_\_.py

book.py # Book class

library.py # Library class (add/search/stats/persist)

core/

iterator.py # LibraryIterator (\_\_iter\_\_/\_\_next\_\_)

utils.py # helpers: string cleaning, id generation

io/

\_\_init\_\_.py

csv\_io.py # import/export CSV

json\_store.py # load/save JSON state

cli/

\_\_init\_\_.py

commands.py # handlers for menu/command routing

data/

books.csv # sample seed data

library.json # runtime state (created at first save)

**5) Feature details & acceptance criteria**

**A) Import books**

* Command: import data/books.csv
* Skips duplicates, reports counts (imported/duplicates/invalid).
* ✔ Acceptance: After import, stats shows correct totals; list displays books.

**B) Add/update/remove**

* add prompts for title/author/genre/pages; sets status="unread".
* update <id> can change status (unread|reading|completed), pages, rating (1–5), add notes.
* remove <id> deletes the book and updates indices.
* ✔ Acceptance: Changes persist to data/library.json after save.

**C) Search & list**

* search title "deep work" (case-insensitive, substring)
* search author "adichie"
* list --mode default|unread|genre
* ✔ Acceptance: Correct results with readable formatting.

**D) Stats**

* stats prints a tuple summary (total, completed, in\_progress) and additional info:
  + Unique authors/genres (from sets)
  + Average rating for completed books
* ✔ Acceptance: Matches the current library state.

**E) Export**

* export data/selection.csv --filter unread
* ✔ Acceptance: Creates CSV with the filtered subset.

**F) Iterator**

* Using for book in library.iter(mode="unread"): yields unread books first.
* ✔ Acceptance: Custom iterator respects selected mode and terminates correctly.

**6) Sample data (data/books.csv)**

title,author,genre,pages

Deep Work,Cal Newport,Productivity,304

The Pragmatic Programmer,Andrew Hunt;David Thomas,Software,352

Half of a Yellow Sun,Chimamanda Ngozi Adichie,Fiction,448

Clean Code,Robert C. Martin,Software,464

Atomic Habits,James Clear,Self-Help,320

Tip: Support multiple authors split by ; (string splitting & normalization).

**7) Deliverables**

1. **Runnable project** in the structure above.
2. **README.md** with:
   * Setup (Python version), how to run (python main.py), command cheatsheet.
   * Data format specs and examples.
3. **Sample dataset** (data/books.csv).
4. **Demonstration script**: a short transcript in README showing a typical session (import → list → search → update → stats → export).
5. **Docstrings & inline comments** explaining iterator, file I/O, and design choices.
6. **(Optional, if time)** Simple unit-style checks in tests/ or a demo.py that exercises core functions.

**8) Suggested time budget (fits 4–6 hours)**

* Scaffolding & models: **60–90 min**
* CSV/JSON I/O: **45–60 min**
* Search, stats, sets/tuples: **45–60 min**
* Iterator modes: **30–45 min**
* CLI & polishing: **60–90 min**

**9) Stretch goals (if you finish early)**

* Fuzzy search (startswith, endswith, or difflib.get\_close\_matches).
* “Smart suggestions” iterator (e.g., shortest unread first; highest-rated unread).
* Tagging system (comma-separated tags → set operations like union/intersection).
* Import from multiple CSVs with conflict reports.
* Basic CSV schema validation with helpful errors.

**10) How each topic is covered (traceability)**

* **Introduction/Getting started**: basic CLI, printing help, variables, control flow.
* **Strings**: normalization, casefold/title, split/join, contains, startswith/endswith.
* **Iterator Protocol**: custom LibraryIterator implementing \_\_iter\_\_/\_\_next\_\_.
* **Lists/Tuples/Sets**: book list; stats tuples; author/genre sets.
* **Dictionaries**: indices and config maps.
* **File Handling**: CSV import, JSON save/load, CSV export.
* **Functions**: small utilities for parsing/validation, pure functions for stats.
* **Modular development**: folders/modules with clear boundaries.
* **Classes & Objects**: Book, Library, iterator objects.