Instructor Runbook
Python Programming (Advanced) – 48 Hours

Delivery format: 12 sessions × 4 instructional hours (48 total)

Prepared for delivery • January 13, 2026

How to use this runbook

This runbook is an hour-by-hour teaching companion for a 48-hour Advanced Python course delivered as 12 sessions of 4 instructional hours each. Each hour block includes: outcomes, instructor talk points, live demo steps, a lab prompt, completion criteria, common pitfalls, optional extensions, and a quick check.

Pacing note for 4-hour sessions: plan one 10-minute break around the midpoint, plus a 3–5 minute micro-break as needed. The hour blocks below are instructional hours.

Prerequisites (expected learner baseline)

- Comfortable with variables, types, data structures (list/dict/set/tuple).
- Can write functions and basic classes; understands control flow and exceptions.
- Can read/write files and work with paths.
- Has completed the 48-hour Basics course or equivalent experience.

Course outcomes (Advanced)

- Apply advanced OOP design (responsibilities, composition, polymorphism) and implement common patterns.
- Build a desktop GUI application (Tkinter or PyQt) connected to application logic.
- Persist and query data using SQLite (and optionally an ORM if time/fit).
- Build and consume RESTful APIs (Flask-based) with clean request/response contracts.
- Perform practical data analysis and visualization with pandas and matplotlib; run a simple regression workflow.
- Write automated tests with pytest and measure basic coverage.
- Package and ship a usable deliverable (requirements + runnable app; optional executable build).
- Prepare for Python Institute PCAP-style exam objectives (advanced practices and code-reading).

Pre-course instructor checklist (Advanced)

- Verify lab/VM access for all learners and that Python 3 is installed and selected correctly in the IDE.
- Confirm package installation is allowed; pre-install or cache required packages if possible.
- Recommended packages (adjust to lab policy): requests, flask, pandas, matplotlib, pytest, coverage (optional: scikit-learn, sqlalchemy, pyinstaller).
- Validate SQLite access (sqlite3 is stdlib) and file permissions for creating .db files in the workspace.
- Prepare a capstone starter repo (or zip) with folder structure: src/, tests/, data/, reports/, README.

- Decide GUI framework to use (default Tkinter for reliability; PyQt only if lab support is confirmed).
- Decide how learners submit work (Git repo preferred) and how you'll do code reviews (pair demos + rubrics).

Session delivery rhythm (recommended)

- Start-of-session (5–10 min): recap + warm-up (predict output / quick fix).
- Teach (10–20 min): concept + concrete examples + a 'why this matters' story.
- Demo (5–10 min): live-code with narration; have learners predict results before running.
- Lab (25–35 min): guided -> independent; circulate; require frequent runs/tests.
- Debrief (5 min): 2–3 learners show solutions; capture 3 rules-of-thumb on a shared doc.

Capstone approach (runs throughout)

Capstone theme: Full-stack Tracker (choose a domain: tasks, inventory, contacts, notes, or expenses).

Minimum deliverables by the end of the course:

- Domain model + service layer (with custom exceptions)
- SQLite persistence layer
- One UI surface: GUI app OR REST API (bonus: both)
- Analytics report (CSV export + at least one chart; optional regression demo)
- Automated tests for core logic (pytest) + basic coverage report
- Packaging: requirements + run instructions; optional executable build

Session overviews (quick schedule)

Session 1 overview (Hours 1–4)

Hour	Focus
1	Advanced kickoff + baseline diagnostic + Git workflow
2	Designing classes from requirements (responsibilities + boundaries)
3	Properties, invariants, and custom exceptions
4	Composition vs inheritance + polymorphism

Session 2 overview (Hours 5–8)

Hour	Focus
5	Pattern: Factory (practical creation + validation)
6	Pattern: Strategy (swap behaviors cleanly)
7	Pythonic class ergonomics (repr/str/equality) + light type hints
8	Checkpoint 1: Domain + service layer milestone

Session 3 overview (Hours 9–12)

Hour	Focus
9	Project structure: packages, imports, and config
10	Logging and error reporting (practical)
11	Context managers + safer file operations
12	Decorators (timing / authorization / validation)

Session 4 overview (Hours 13–16)

Hour	Focus
13	HTTP client work: requests + JSON contracts
14	Security basics (appropriate scope)
15	Capstone planning workshop (choose delivery path)
16	Checkpoint 2: Persistence-ready core + JSON save/load

Session 5 overview (Hours 17–20)

Hour	Focus
17	GUI fundamentals (Tkinter/ttk): event loop, widgets, callbacks
18	Layout and usability: grid/pack, spacing, resizing
19	Forms + validation feedback patterns
20	Record list UI: Listbox/Treeview + selection events

Session 6 overview (Hours 21–24)

Hour	Focus
21	CRUD wiring in GUI: update and delete workflows
22	GUI architecture: controller separation + avoiding callback spaghetti
23	GUI mini-project sprint: persistence + polish
24	Checkpoint 3: GUI milestone demo

Session 7 overview (Hours 25–28)

Hour	Focus
25	From objects to tables: schema thinking + repository idea
26	sqlite3 CRUD with parameterized queries
27	Row mapping: converting rows to objects
28	Transactions + integrity + initialization patterns

Session 8 overview (Hours 29–32)

Hour	Focus
29	Optional ORM slice (SQLAlchemy) OR deeper SQL practice
30	Integrate DB into the app (service layer uses repository)
31	Search/filter + pagination patterns (practical)
32	Checkpoint 4: Data-driven app milestone

Session 9 overview (Hours 33–36)

Hour	Focus
33	REST fundamentals + Flask app setup
34	CRUD endpoints for your main resource
35	Serialization + validation (manual, consistent)
36	App structure: blueprints (optional) + dependency wiring

Session 10 overview (Hours 37–40)

Hour	Focus
37	API security basics: API key gate + safe storage
38	Consuming your own API: Python client functions
39	Integration choice: GUI talks to API OR parallel UI/API
40	Checkpoint 5: API milestone demo

Session 11 overview (Hours 41–44)

Hour	Focus
41	Pandas essentials: loading, cleaning, summarizing
42	Visualization with matplotlib: charts you can ship
43	Regression demo (practical, limited scope)
44	Integrate analytics into capstone (report feature)

Session 12 overview (Hours 45–48)

Hour	Focus
45	Testing with pytest: unit tests for service layer
46	Coverage + edge cases + lightweight integration test
47	Packaging and delivery: requirements + runnable app + optional executable
48	Final: capstone demo + certification-style review + retrospective

48-hour hour-by-hour lesson blocks

Each hour block includes outcomes, instructor talk points, demo steps, lab prompt, completion criteria, pitfalls, optional extensions, and a quick check.

Session 1 (Hours 1–4)

Hour 1: Advanced kickoff + baseline diagnostic + Git workflow

Outcomes

- Confirm environment and required packages.
- Set a working Git workflow for the course repo.
- Assess baseline skills quickly.

Instructor talk points (10–20 min)

- Course workflow: lectures + labs + capstone milestones.
- Advanced expectations: readability, structure, and testing mindset.
- Git basics for the course: clone/pull/commit/push; branching optional.

Live demo (5–10 min)

1. Create repo folder and initialize Git.
2. Create and run 'diagnostic.py' (provided or quick skeleton).
3. Commit initial state: 'chore: initial setup'.

Hands-on lab (25–35 min)

Lab: Setup + Diagnostic

- Create a project folder with src/, tests/, data/, reports/.

- Initialize Git and make the first commit.

- Run a short diagnostic script that exercises: a class, file read/write, exception, and a dict.

- Record one personal goal for the week (e.g., 'get comfortable with Flask routing').

Completion criteria

- Repo exists with standard folders.
- Diagnostic runs successfully.
- At least one commit pushed or saved locally.

Common pitfalls to watch for

- Wrong interpreter/venv selected.
- Forgetting to commit frequently.
- Workspace path issues in the lab VM.

Optional extensions (stay in Advanced scope)

- Create a venv and freeze requirements (if allowed).
- Add a .gitignore (pycache, venv, .db, reports).

Quick check / exit ticket

Quick check: What's the difference between commit and push?

Hour 2: Designing classes from requirements (responsibilities + boundaries)

Outcomes

- Translate requirements into classes and responsibilities.
- Identify where validation and business rules should live.

Instructor talk points (10–20 min)

- Responsibility-driven design: what does the object know/do?
- Keep UI separate from core logic.
- Small, testable functions and methods.

Live demo (5–10 min)

4. Sketch 3–5 classes for a tracker domain.
5. Show a simple service layer function that uses the classes.

Hands-on lab (25–35 min)

Lab: Domain modeling

- Pick a capstone theme.

- Write a short requirements list (5–8 bullets).

- Define 3–5 classes with attributes + methods (in comments or a simple diagram).

- Implement at least one class with __init__ and one behavior method.

Completion criteria

- At least one working class implemented.
- Clear separation between model and UI code.

Common pitfalls to watch for

- Putting I/O (input/print) inside domain objects.
- Overcomplicated inheritance early.

Optional extensions (stay in Advanced scope)

- Add type hints for model attributes.
- Add __str__ for friendly display.

Quick check / exit ticket

Quick check: Where should validation live: UI layer, service layer, or model?

Hour 3: Properties, invariants, and custom exceptions

Outcomes

- Use @property for validation.
- Define custom exceptions for domain errors.

Instructor talk points (10–20 min)

- Properties as controlled access.
- Invariants (rules that must always be true).
- Custom exceptions vs returning None/False.

Live demo (5–10 min)

6. Live-code a model with a validated property.
7. Show raising and catching a custom ValidationError.

Hands-on lab (25–35 min)

Lab: Validation + exceptions

- Add at least one validation rule to your model (e.g., non-empty name, positive amount).

- Define custom exceptions: ValidationError, NotFoundError.

- Update one service function to raise these exceptions.

Completion criteria

- Invalid inputs trigger a clear exception.
- Caller handles error with a user-friendly message (later in UI).

Common pitfalls to watch for

- Catching Exception broadly and hiding the cause.
- Validation rules that are too strict or inconsistent.

Optional extensions (stay in Advanced scope)

- Add an error message attribute to your exception.
- Add a helper to normalize user input (strip/case).

Quick check / exit ticket

Quick check: Why is catching ValueError specifically better than catching everything?

Hour 4: Composition vs inheritance + polymorphism

Outcomes

- Choose composition or inheritance appropriately.
- Use polymorphism to simplify branching.

Instructor talk points (10–20 min)

- Inheritance is for 'is-a'; composition for 'has-a'.
- Polymorphism via shared method names (duck typing).

Live demo (5–10 min)

8. Refactor a messy if/elif into two classes with the same method name.
9. Show a composed object holding another object.

Hands-on lab (25–35 min)

Lab: Refactor exercise

- Start with a feature implemented using if/elif branching.

- Refactor into 2 classes that share a method (e.g., export formats: to_text / to_dict).

- Demonstrate calling them through the same interface.

Completion criteria

- Refactor reduces branching.
- Feature still works and is easier to read.

Common pitfalls to watch for

- Overengineering: too many tiny classes.
- Mixing inheritance and composition without reason.

Optional extensions (stay in Advanced scope)

- Add a third implementation (e.g., CSV exporter).

Quick check / exit ticket

Quick check: Name one scenario where composition is safer than inheritance.

Session 2 (Hours 5–8)

Hour 5: Pattern: Factory (practical creation + validation)

Outcomes

- Implement a simple factory function/class.
- Centralize object creation and normalization.

Instructor talk points (10–20 min)

- Why factories: consistent creation, fewer duplicated rules.
- Factory returning fully-valid objects.

Live demo (5–10 min)

10. Build RecordFactory.from_dict() for JSON input.
11. Show error on missing fields.

Hands-on lab (25–35 min)

Lab: Factory implementation

- Implement a factory that creates your model from a dict (like JSON input).

- Add required-field checks and defaults.

- Use it to load 2 sample records from hard-coded dicts.

Completion criteria

- Factory creates valid objects or raises ValidationError.
- Loaded objects can be printed/displayed.

Common pitfalls to watch for

- Trying to serialize custom objects directly without conversion.
- Silent defaults hide mistakes.

Optional extensions (stay in Advanced scope)

- Add from_user_input() factory that normalizes fields.

Quick check / exit ticket

Quick check: What’s the benefit of centralizing creation logic?

Hour 6: Pattern: Strategy (swap behaviors cleanly)

Outcomes

- Define strategies as callables or classes.
- Use strategies to avoid duplicated sorting/filtering logic.

Instructor talk points (10–20 min)

- Strategy pattern: selecting an algorithm at runtime.
- Keep it lightweight in Python (functions work!).

Live demo (5–10 min)

12. Demo: choose different sort keys based on user selection.
13. Demo: validation strategies for different record types.

Hands-on lab (25–35 min)

Lab: Strategy selection

- Implement two strategies (e.g., sort by name vs by created_date; filter active vs all).

- Wire a menu option to choose strategy and display results.

Completion criteria

- Strategy changes behavior without rewriting core code.
- Results are clearly different and correct.

Common pitfalls to watch for

- Strategies that depend on global variables.
- Hard-coded branching still dominates.

Optional extensions (stay in Advanced scope)

- Add a third strategy; store strategies in a dict for lookup.

Quick check / exit ticket

Quick check: How is a strategy different from an if/elif chain?

Hour 7: Pythonic class ergonomics (repr/str/equality) + light type hints

Outcomes

- Improve debuggability with __repr__ and __str__.

- Use light type hints for readability.

Instructor talk points (10–20 min)

- __repr__ for developers, __str__ for users.
- Type hints help humans and tools.

Live demo (5–10 min)

14. Add __repr__ to model and show how it helps debugging.
15. Add type hints to a couple methods.

Hands-on lab (25–35 min)

Lab: Make your model pleasant to work with

- Add __repr__ and __str__ to your main model.

- Add type hints to 3 methods/functions.

- Ensure output is readable in logs and UI.

Completion criteria

- Objects print cleanly.
- Hints are correct and don't break runtime.

Common pitfalls to watch for

- Overcomplicated typing (keep it simple).
- Returning None unexpectedly.

Optional extensions (stay in Advanced scope)

- Add a to_dict() method for serialization.

Quick check / exit ticket

Quick check: When would you look at __repr__ output vs __str__ output?

Hour 8: Checkpoint 1: Domain + service layer milestone

Outcomes

- Deliver a stable core package that can be reused by GUI/API layers.

Instructor talk points (10–20 min)

- Checkpoint expectations and rubric.
- Testing approach: run small examples, then edge cases.

Live demo (5–10 min)

16. Demo: how you'll grade quickly (run script, try invalid input, try missing record).

Hands-on lab (25–35 min)

Checkpoint 1 (45–60 min)

Build: Core Tracker Library

Required:

- 2+ model classes OR 1 model + 1 helper class

- service layer functions (add/list/search/update/delete)

- custom exceptions

- serialization helpers (to_dict at minimum)

Deliverable: src/ core runs via a small demo script.

Completion criteria

- Meets requirements and runs end-to-end.
- Errors are meaningful and caught appropriately.

Common pitfalls to watch for

- Global state everywhere.
- No separation between model and UI.
- No handling for missing records.

Optional extensions (stay in Advanced scope)

- Add: simple CLI wrapper for the service layer.

Quick check / exit ticket

Quick check: What part of your design are you happiest with, and why?

Day 2

Session 3 (Hours 9–12)

Hour 9: Project structure: packages, imports, and config

Outcomes

- Organize code into a package.
- Avoid common import pitfalls.

Instructor talk points (10–20 min)

- src layout basics.
- Relative vs absolute imports (keep pragmatic).
- Config via constants and env vars (light).

Live demo (5–10 min)

17. Move model/service code into src/ package and adjust imports.
18. Run as module (python -m) if helpful.

Hands-on lab (25–35 min)

Lab: Restructure

- Create src/tracker/ package with __init__.py.

- Move models.py and services.py into the package.

- Update imports and confirm the demo script still works.

Completion criteria

- Project runs after restructure.
- Imports are clean and consistent.

Common pitfalls to watch for

- Circular imports.
- Running the wrong entry file.
- Naming a module the same as a stdlib package.

Optional extensions (stay in Advanced scope)

- Add a config.py for constants (DATA_DIR, DB_PATH).

Quick check / exit ticket

Quick check: What usually causes 'ModuleNotFoundError' in student projects?

Hour 10: Logging and error reporting (practical)

Outcomes

- Use the logging module for diagnostics.
- Log errors without spamming users.

Instructor talk points (10–20 min)

- logging levels: DEBUG/INFO/WARNING/ERROR.
- Formatting and file logging.
- Do not print stack traces to end users.

Live demo (5–10 min)

19. Add a logger to services; log ValidationError and NotFoundError.
20. Show log file output.

Hands-on lab (25–35 min)

Lab: Add logging

- Configure logging to write to logs/app.log.

- Add at least 3 log calls in your service layer.

- Trigger one error and confirm it appears in the log.

Completion criteria

- Log file created and populated.
- User-facing messages remain friendly.

Common pitfalls to watch for

- Logging inside tight loops (noise).
- Forgetting to create logs/ directory.

Optional extensions (stay in Advanced scope)

- Add timestamp + module name formatting.

Quick check / exit ticket

Quick check: When should you log at WARNING vs ERROR?

Hour 11: Context managers + safer file operations

Outcomes

- Use context managers consistently.
- Write safe save patterns (write temp then replace).

Instructor talk points (10–20 min)

- Why 'with' matters beyond files.
- Atomic-ish saves to avoid corrupted files.

Live demo (5–10 min)

21. Demo: write JSON to temp file then replace original.
22. Show pathlib usage.

Hands-on lab (25–35 min)

Lab: Safe save utility

- Implement a save_json_safe(path, data) helper.

- Use it to save your tracker data to a JSON file.

- Simulate a failure (raise exception mid-write) and discuss what can go wrong without safe saves.

Completion criteria

- Saves use with + safe pattern.
- Files are not left half-written.

Common pitfalls to watch for

- Overwriting a good file with bad data.
- Assuming working directory is stable.

Optional extensions (stay in Advanced scope)

- Add a timestamped backup file before replace.

Quick check / exit ticket

Quick check: What’s one advantage of writing to a temp file first?

Hour 12: Decorators (timing / authorization / validation)

Outcomes

- Write and apply a simple decorator.
- Use a decorator to reduce duplicated logic.

Instructor talk points (10–20 min)

- Decorator anatomy: wrapper(*args, **kwargs).
- Keep decorators small and readable.

Live demo (5–10 min)

23. Demo: @timed decorator for service methods.
24. Demo: @requires_api_key (toy example).

Hands-on lab (25–35 min)

Lab: Decorator practice

- Implement @timed that logs execution time.

- Apply it to 2 service functions.

- Verify timing logs appear when functions are called.

Completion criteria

- Decorator works and does not change function behavior.
- Logs show timing output.

Common pitfalls to watch for

- Forgetting to return function result from wrapper.
- Breaking function signatures accidentally.

Optional extensions (stay in Advanced scope)

- Preserve function name with functools.wraps (optional).

Quick check / exit ticket

Quick check: What happens if your wrapper forgets to return the result?

Session 4 (Hours 13–16)

Hour 13: HTTP client work: requests + JSON contracts

Outcomes

- Consume a JSON API with requests.
- Handle network failures gracefully.

Instructor talk points (10–20 min)

- GET vs POST at a high level.
- JSON parsing and error cases.
- Timeouts and status codes (practical).

Live demo (5–10 min)

25. Demo: call a public sample API (or provided local endpoint).
26. Show try/except for requests exceptions.

Hands-on lab (25–35 min)

Lab: API consumer

- Write a small script that calls an HTTP endpoint (public or instructor-provided).

- Parse JSON and display key fields.

- Handle non-200 status codes with a friendly message.

Completion criteria

- Successful request and parsed output.
- Graceful handling of failure cases.

Common pitfalls to watch for

- No timeout set (hangs).
- Assuming JSON structure without checks.

Optional extensions (stay in Advanced scope)

- Add a query parameter; add retries (simple: loop 3 times).

Quick check / exit ticket

Quick check: What’s the difference between a timeout and a retry?

Hour 14: Security basics (appropriate scope)

Outcomes

- Keep secrets out of code.
- Use basic hashing for stored secrets (conceptually).

Instructor talk points (10–20 min)

- Environment variables for secrets.
- Hashing vs encryption (high-level).
- Don’t roll your own crypto; keep it simple.

Live demo (5–10 min)

27. Demo: read API key from env var.
28. Demo: hash a string with hashlib (sha256) for comparison-only use.

Hands-on lab (25–35 min)

Lab: Secure-ish configuration

- Add support for an API key read from an environment variable.

- Implement a simple 'admin' action gated by the key.

- Do not hard-code the key in code or commits.

Completion criteria

- Key is loaded from env/config.
- Admin action is gated and logged.

Common pitfalls to watch for

- Accidentally committing secrets.
- Confusing hashing with encryption.

Optional extensions (stay in Advanced scope)

- Add a .env template file (without real secrets).

Quick check / exit ticket

Quick check: Why is hashing not reversible (and why is that useful)?

Hour 15: Capstone planning workshop (choose delivery path)

Outcomes

- Lock capstone scope and milestones.
- Choose GUI-first or API-first implementation path.

Instructor talk points (10–20 min)

- Milestones: core -> persistence -> UI -> analytics -> tests -> package.
- Definition of done for each milestone.

Live demo (5–10 min)

29. Demo: sample folder structure and README skeleton.
30. Show a simple issue checklist.

Hands-on lab (25–35 min)

Lab: Capstone plan

- Write a one-page plan (bullets) with:

- scope (features)

- data model summary

- chosen UI surface(s)

- persistence plan

- analytics idea

- test plan (what will you test)

- Create a README skeleton with run instructions placeholders.

Completion criteria

- Plan exists and is realistic.
- README skeleton created.

Common pitfalls to watch for

- Scope too big.
- No plan for error handling or tests.

Optional extensions (stay in Advanced scope)

- Add a 'stretch goals' section to keep MVP safe.

Quick check / exit ticket

Quick check: What feature is your MVP, and what is your biggest risk?

Hour 16: Checkpoint 2: Persistence-ready core + JSON save/load

Outcomes

- Deliver a core package that can reliably save/load state and is ready for DB integration.

Instructor talk points (10–20 min)

- Checkpoint grading focus: correctness + structure + error handling + logging.

Live demo (5–10 min)

31. Demo: your 'fast grade' procedure: run, load missing file, save, reload, trigger validation.

Hands-on lab (25–35 min)

Checkpoint 2 (45–60 min)

Build: Persistence-ready Core

Required:

- package layout under src/

- JSON save/load (safe save recommended)

- logging to file

- custom exceptions still used

Deliverable: demo script that shows save -> restart -> load.

Completion criteria

- State persists across runs.
- Errors are handled without crashing.
- Logs contain meaningful entries.

Common pitfalls to watch for

- Dumping objects directly to JSON.
- Corrupting JSON and not handling JSONDecodeError.

Optional extensions (stay in Advanced scope)

- Add a migration step (version field in JSON) if time.

Quick check / exit ticket

Quick check: What’s one design decision you would change if you started over?

Day 3

Session 5 (Hours 17–20)

Hour 17: GUI fundamentals (Tkinter/ttk): event loop, widgets, callbacks

Outcomes

- Create a window and basic widgets.
- Wire callbacks to service functions.

Instructor talk points (10–20 min)

- GUI is event-driven; avoid blocking calls.
- Basic widgets: Label, Entry, Button.
- Separate UI from service logic.

Live demo (5–10 min)

32. Demo: minimal GUI that adds a record using the service layer.
33. Show messagebox feedback.

Hands-on lab (25–35 min)

Lab: Hello GUI + Add form

- Create a Tkinter app with:

- fields for a record

- Add button

- status message area

- On Add: validate via service layer and show success/error.

Completion criteria

- GUI launches reliably.
- Add action works and shows feedback.

Common pitfalls to watch for

- Putting business logic inside callbacks.
- Forgetting mainloop(), widget not displayed due to layout issues.

Optional extensions (stay in Advanced scope)

- Add keyboard 'Enter' binding to submit.

Quick check / exit ticket

Quick check: What does mainloop() do?

Hour 18: Layout and usability: grid/pack, spacing, resizing

Outcomes

- Build a clean layout with consistent spacing.
- Handle resizing responsibly.

Instructor talk points (10–20 min)

- grid() for forms; sticky + padding.
- Using frames to group UI.
- Simple usability checklist.

Live demo (5–10 min)

34. Refactor messy UI into 2 frames (form + list).
35. Demonstrate columnconfigure weight.

Hands-on lab (25–35 min)

Lab: Improve layout

- Rebuild the UI using frames.

- Add consistent padding.

- Ensure window resizing doesn't break the layout.

Completion criteria

- UI is readable and aligned.
- Resizing is acceptable.

Common pitfalls to watch for

- Mixing pack and grid in same container.
- Widgets overlap due to missing sticky/padding.

Optional extensions (stay in Advanced scope)

- Add a toolbar frame with 2–3 buttons (Add, Delete, Refresh).

Quick check / exit ticket

Quick check: Why shouldn't you mix pack and grid in the same parent?

Hour 19: Forms + validation feedback patterns

Outcomes

- Validate input before saving.
- Display validation errors clearly.

Instructor talk points (10–20 min)

- Inline validation vs messagebox.
- Keeping error messages actionable.
- Disable/enable buttons when needed.

Live demo (5–10 min)

36. Demo: highlight invalid field + show message.
37. Show trimming/normalizing input before validate.

Hands-on lab (25–35 min)

Lab: Add validation UI

- Add required field checks (non-empty).

- Show validation feedback without crashing.

- Ensure errors from service layer are surfaced cleanly.

Completion criteria

- Invalid entries show clear feedback.
- Valid entries save successfully.

Common pitfalls to watch for

- Silent failure (button does nothing).
- Stack trace shown to user.

Optional extensions (stay in Advanced scope)

- Add a 'Clear form' button.

Quick check / exit ticket

Quick check: What is a good validation message vs a bad one?

Hour 20: Record list UI: Listbox/Treeview + selection events

Outcomes

- Display records in a list/table.
- React to selection to show details.

Instructor talk points (10–20 min)

- Listbox vs Treeview; when to use which.
- Selection callbacks.
- Refreshing list safely.

Live demo (5–10 min)

38. Demo: Treeview showing records; select row -> populate details pane.

Hands-on lab (25–35 min)

Lab: List + details

- Add a Treeview (or Listbox) that lists records.

- Add a details area that shows the selected record.

- Add a Refresh button that reloads from your in-memory data.

Completion criteria

- List populates and updates.
- Selecting a record shows details.

Common pitfalls to watch for

- Not clearing old rows before repopulating.
- Crashes when no selection exists.

Optional extensions (stay in Advanced scope)

- Add sorting by clicking a column header (optional).

Quick check / exit ticket

Quick check: What should happen if the user clicks Refresh with no records?

Session 6 (Hours 21–24)

Hour 21: CRUD wiring in GUI: update and delete workflows

Outcomes

- Implement update and delete from the GUI.
- Confirm destructive actions.

Instructor talk points (10–20 min)

- Editing pattern: select -> load into form -> save update.
- Confirm delete with messagebox.

Live demo (5–10 min)

39. Demo: update flow and delete confirmation.
40. Show handling NotFoundError gracefully.

Hands-on lab (25–35 min)

Lab: Update/Delete

- Implement Update using selected record.

- Implement Delete with confirmation.

- After each change, refresh the list automatically.

Completion criteria

- Update and delete work reliably.
- UI stays in sync after changes.

Common pitfalls to watch for

- Using list index as ID and breaking when list order changes.
- Deleting while iterating.

Optional extensions (stay in Advanced scope)

- Add undo (simple: last deleted stored in memory) if time.

Quick check / exit ticket

Quick check: Why is a stable ID better than relying on list index?

Hour 22: GUI architecture: controller separation + avoiding callback spaghetti

Outcomes

- Keep UI code maintainable.
- Separate widget creation from event handlers.

Instructor talk points (10–20 min)

- Thin UI callbacks call service layer.
- Store app state cleanly.
- Use a controller class to hold references.

Live demo (5–10 min)

41. Refactor to an App class with methods: build_ui(), refresh(), on_add(), on_delete().

Hands-on lab (25–35 min)

Lab: Refactor GUI

- Create an App class that builds UI and owns callbacks.

- Move service calls into a separate module if not already.

- Confirm behavior remains the same after refactor.

Completion criteria

- Code is easier to read.
- Functionality preserved.

Common pitfalls to watch for

- Over-refactor midstream and break working code.
- Circular imports between UI and service modules.

Optional extensions (stay in Advanced scope)

- Add a status bar logger (shows last action).

Quick check / exit ticket

Quick check: What's one sign your GUI code needs refactoring?

Hour 23: GUI mini-project sprint: persistence + polish

Outcomes

- Add persistence and ‘real app’ polish to the GUI milestone.

Instructor talk points (10–20 min)

- Load data on startup; save after changes.
- Graceful error messages for file/JSON problems.

Live demo (5–10 min)

42. Demo: start app -> load -> add -> save -> restart -> load.

Hands-on lab (25–35 min)

Lab: GUI milestone sprint

- Add load on startup and save after CRUD.

- Add menu items (File -> Export, Help -> About) OR a simple help dialog.

- Add basic keyboard shortcuts (Ctrl+S to save) if framework supports easily.

Completion criteria

- GUI persists data across runs.
- User feedback is clear.

Common pitfalls to watch for

- Forgetting to save after update/delete.
- Corrupt JSON causes crash.

Optional extensions (stay in Advanced scope)

- Add an export-to-CSV button (used later for analytics).

Quick check / exit ticket

Quick check: What’s one usability improvement you added that you’d want in a real tool?

Hour 24: Checkpoint 3: GUI milestone demo

Outcomes

- Deliver a working GUI CRUD app wired to your service layer.

Instructor talk points (10–20 min)

- Grading focus: UI works, CRUD complete, errors handled, code organization.

Live demo (5–10 min)

43. Demo: fast grading checklist run.

Hands-on lab (25–35 min)

Checkpoint 3 (45–60 min)

Deliver:

- GUI launches reliably

- CRUD works through UI

- load/save works (JSON ok)

- clean separation (ui/ vs core)

Short demo: add -> list -> update -> delete -> restart -> load.

Completion criteria

- End-to-end GUI workflow demonstrated.
- No crashes in typical use.

Common pitfalls to watch for

- UI-only validation and no service validation.
- IDs unstable causing wrong updates/deletes.

Optional extensions (stay in Advanced scope)

- Add search/filter UI.

Quick check / exit ticket

Quick check: If you had one more hour, what would you improve first in your GUI?

Day 4

Session 7 (Hours 25–28)

Hour 25: From objects to tables: schema thinking + repository idea

Outcomes

- Map your model to a relational table.
- Define a repository interface.

Instructor talk points (10–20 min)

- Table design: columns, types, primary key.
- CRUD queries you will need.
- Repository isolates DB details.

Live demo (5–10 min)

44. Demo: simple schema for your records.
45. Show a repository method signature: add(record) -> id.

Hands-on lab (25–35 min)

Lab: Design schema

- Draft a table schema for your main record.

- Decide on a stable ID type (INTEGER PRIMARY KEY or TEXT).

- Write a repository interface (class with method stubs) with CRUD methods.

Completion criteria

- Schema draft exists.
- Repository methods defined.

Common pitfalls to watch for

- Trying to store nested objects without plan.
- No stable ID.

Optional extensions (stay in Advanced scope)

- Add a second table for a related entity (optional).

Quick check / exit ticket

Quick check: Why is a repository useful even for small projects?

Hour 26: sqlite3 CRUD with parameterized queries

Outcomes

- Connect to SQLite and run CRUD safely.
- Use parameterized queries to avoid injection.

Instructor talk points (10–20 min)

- sqlite3 connection, cursor, commit.
- CREATE TABLE IF NOT EXISTS.
- INSERT/SELECT/UPDATE/DELETE with ? placeholders.

Live demo (5–10 min)

46. Demo: create db file; create table; insert one row; select rows.

Hands-on lab (25–35 min)

Lab: First SQLite integration

- Create a db file in data/ (e.g., tracker.db).

- Create the table.

- Implement repo.add() and repo.list_all().

- Print results to confirm.

Completion criteria

- DB file created.
- Insert and list works.
- Queries use parameters.

Common pitfalls to watch for

- Forgetting commit().
- String formatting SQL (unsafe).
- DB path issues.

Optional extensions (stay in Advanced scope)

- Add a simple search query (WHERE name LIKE ?).

Quick check / exit ticket

Quick check: What does the ? placeholder do in sqlite3?

Hour 27: Row mapping: converting rows to objects

Outcomes

- Map DB rows back into your model objects.
- Keep mapping code centralized.

Instructor talk points (10–20 min)

- Row tuples/dicts; column ordering.
- Factory from_row() pattern.
- Handling null/optional fields.

Live demo (5–10 min)

47. Demo: repo.get_by_id returns a model object.
48. Show NotFoundError when no row.

Hands-on lab (25–35 min)

Lab: Implement get/update/delete

- Implement repo.get_by_id.

- Implement repo.update.

- Implement repo.delete.

- Ensure missing IDs raise NotFoundError at the service layer.

Completion criteria

- CRUD repo methods complete.
- Service layer handles missing records.

Common pitfalls to watch for

- Mismatched column ordering.
- Update query missing WHERE clause (oops!).

Optional extensions (stay in Advanced scope)

- Add 'updated_at' timestamp column (optional).

Quick check / exit ticket

Quick check: What’s the fastest way to confirm your UPDATE query worked?

Hour 28: Transactions + integrity + initialization patterns

Outcomes

- Use transactions for grouped changes.
- Initialize schema safely on startup.

Instructor talk points (10–20 min)

- Connection context manager.
- Foreign keys (mention lightly).
- Idempotent init: safe to run repeatedly.

Live demo (5–10 min)

49. Demo: init_db() function and calling it once at startup.
50. Demo: transaction for multiple inserts.

Hands-on lab (25–35 min)

Lab: Hardening DB layer

- Write init_db(db_path) that sets up schema.

- Ensure it runs safely if the table already exists.

- Add basic constraints (NOT NULL for required fields).

Completion criteria

- DB initializes reliably.
- Constraints prevent bad data.

Common pitfalls to watch for

- Schema drift and manual edits.
- Assuming foreign keys on (SQLite needs PRAGMA).

Optional extensions (stay in Advanced scope)

- Add a seed_data() function for demos.

Quick check / exit ticket

Quick check: Why should init_db be safe to run multiple times?

Session 8 (Hours 29–32)

Hour 29: Optional ORM slice (SQLAlchemy) OR deeper SQL practice

Outcomes

- (Optional) Understand what an ORM does.
- OR: practice advanced SQL queries safely.

Instructor talk points (10–20 min)

- If using ORM: models + session basics.
- If not: JOINs and aggregates (kept small).

Live demo (5–10 min)

51. Demo option A: a tiny SQLAlchemy model and one query.
52. Demo option B: COUNT(*) group by status.

Hands-on lab (25–35 min)

Lab (choose one)

A) ORM mini-lab (if supported): create one SQLAlchemy model and list rows.

B) SQL mini-lab: add a query that counts records by a category/status and prints a summary.

Completion criteria

- One optional path completed.
- Learner can explain tradeoffs at a high level.

Common pitfalls to watch for

- Installing SQLAlchemy in locked-down labs.
- Overcomplicating by switching approaches mid-course.

Optional extensions (stay in Advanced scope)

- Add a simple migration note to README.

Quick check / exit ticket

Quick check: What's one advantage and one disadvantage of an ORM?

Hour 30: Integrate DB into the app (service layer uses repository)

Outcomes

- Switch the service layer from in-memory/JSON to SQLite repository.
- Keep UI/API code unchanged where possible.

Instructor talk points (10–20 min)

- Dependency injection concept (pass repo into service).
- Minimize refactors by keeping same service interface.

Live demo (5–10 min)

53. Demo: service = TrackerService(repo=SQLiteRepo(...))
54. Show UI still works after swap.

Hands-on lab (25–35 min)

Lab: Wire it up

- Replace JSON persistence with SQLite persistence.

- Keep UI logic mostly unchanged.

- Confirm: add/list/update/delete all work and persist across restarts.

Completion criteria

- App persists via SQLite.
- UI/API still functions.

Common pitfalls to watch for

- Two sources of truth (JSON + DB) causing confusion.
- Forgetting to refresh UI after DB changes.

Optional extensions (stay in Advanced scope)

- Keep export/import using CSV/JSON as an extra feature.

Quick check / exit ticket

Quick check: What interface stayed the same when you swapped storage?

Hour 31: Search/filter + pagination patterns (practical)

Outcomes

- Add a search feature backed by SQL.
- Implement basic paging to keep UI responsive.

Instructor talk points (10–20 min)

- LIKE queries with parameters.
- LIMIT/OFFSET basics.
- UI: refresh pattern after search.

Live demo (5–10 min)

55. Demo: repo.search(q, limit, offset).
56. Demo: next/prev buttons in UI (optional).

Hands-on lab (25–35 min)

Lab: Search feature

- Add a search box in GUI OR query param in a small script.

- Implement a repository search method.

- (Optional) Add paging with limit 20 and Next/Prev.

Completion criteria

- Search returns correct results.
- No SQL injection risk.

Common pitfalls to watch for

- Building SQL with string concatenation.
- Forgetting wildcards in LIKE.

Optional extensions (stay in Advanced scope)

- Add sorting options (order by name/date).

Quick check / exit ticket

Quick check: Where should wildcard '%' be added: in SQL string or the parameter value?

Hour 32: Checkpoint 4: Data-driven app milestone

Outcomes

- Deliver an app that persists data in SQLite and supports key workflows.

Instructor talk points (10–20 min)

- Grading focus: CRUD + persistence + organization + safe SQL.

Live demo (5–10 min)

57. Demo: fast-grade checklist.

Hands-on lab (25–35 min)

Checkpoint 4 (45–60 min)

Deliver:

- SQLite repo with CRUD

- init_db on startup

- app uses DB as source of truth

- search works (basic)

Short demo: create -> close -> reopen -> verify data persists; run search.

Completion criteria

- SQLite persistence proven.
- Queries parameterized.
- No crashes on common errors.

Common pitfalls to watch for

- Update/delete affects wrong record due to ID mistakes.
- Schema mismatch with code.

Optional extensions (stay in Advanced scope)

- Add a second table (stretch) and show one JOIN.

Quick check / exit ticket

Quick check: What's the most common DB bug you hit today?

Day 5

Session 9 (Hours 33–36)

Hour 33: REST fundamentals + Flask app setup

Outcomes

- Create a Flask app and a health endpoint.
- Explain routes/methods/status codes.

Instructor talk points (10–20 min)

- REST basics: resources, verbs, status codes.
- Flask app structure and run modes.

Live demo (5–10 min)

58. Demo: /health returns JSON.
59. Demo: curl a GET request.

Hands-on lab (25–35 min)

Lab: Flask starter

- Create api/app.py with Flask.

- Add /health that returns {'status':'ok'}.

- Run and test with browser or curl.

Completion criteria

- App runs and responds.
- Learner can explain GET vs POST.

Common pitfalls to watch for

- Port conflicts.
- Debug mode assumptions in shared labs.

Optional extensions (stay in Advanced scope)

- Add a /version endpoint from a constant.

Quick check / exit ticket

Quick check: What HTTP status code would you return for a successful create?

Hour 34: CRUD endpoints for your main resource

Outcomes

- Implement GET/POST/PUT/DELETE for records.
- Return proper JSON and status codes.

Instructor talk points (10–20 min)

- Route design: /records and /records/<id>.
- Parsing JSON body safely.
- Returning 400 vs 404.

Live demo (5–10 min)

60. Demo: POST create; GET list; GET by id; show 404 path.

Hands-on lab (25–35 min)

Lab: Build CRUD API

- Implement:

- GET /records

- POST /records

- GET /records/<id>

- PUT /records/<id>

- DELETE /records/<id>

- Use your service layer + SQLite repo behind the API.

Completion criteria

- All endpoints work with correct responses.
- Errors return JSON with message.

Common pitfalls to watch for

- Not validating JSON body.
- Returning 200 for everything.
- Mixing int/string IDs.

Optional extensions (stay in Advanced scope)

- Add query param filtering: /records?q=...

Quick check / exit ticket

Quick check: When should you return 400 vs 404?

Hour 35: Serialization + validation (manual, consistent)

Outcomes

- Standardize request/response shapes.
- Keep validation in one place.

Instructor talk points (10–20 min)

- to_dict/from_dict reuse.
- Error payload format: {'error': '...'}
- Don’t duplicate validation rules in routes.

Live demo (5–10 min)

61. Demo: central parse_record(payload) that raises ValidationError.
62. Show error handling wrapper.

Hands-on lab (25–35 min)

Lab: Clean API contracts

- Create a serializer module for record payloads.

- Ensure all errors use a consistent JSON shape.

- Add 3 negative tests manually: missing field, bad type, missing ID.

Completion criteria

- Contract is consistent across endpoints.
- Invalid inputs return helpful errors.

Common pitfalls to watch for

- Route handlers become huge.
- Different error shapes per endpoint.

Optional extensions (stay in Advanced scope)

- Add a response wrapper that includes request_id (simple random token).

Quick check / exit ticket

Quick check: Why should all API errors follow a consistent format?

Hour 36: App structure: blueprints (optional) + dependency wiring

Outcomes

- Keep Flask code maintainable as it grows.
- Wire repos/services cleanly.

Instructor talk points (10–20 min)

- Simple application factory pattern (pragmatic).
- Where to create DB connection/repo.

Live demo (5–10 min)

63. Demo: create_app() that initializes repo and registers routes.
64. Keep it simple (no overframeworking).

Hands-on lab (25–35 min)

Lab: Refactor API structure

- Move routes into a module (api/routes.py).

- Create a create_app() function.

- Ensure you can still run the app and hit /health and /records.

Completion criteria

- App still runs after refactor.
- Services/repos constructed predictably.

Common pitfalls to watch for

- Circular imports between app and routes.
- Global singletons with hidden state.

Optional extensions (stay in Advanced scope)

- Add a config value for DB path.

Quick check / exit ticket

Quick check: What's the risk of creating your repo/service at import time?

Session 10 (Hours 37–40)

Hour 37: API security basics: API key gate + safe storage

Outcomes

- Add a simple API key check for write operations.
- Avoid storing secrets in code.

Instructor talk points (10–20 min)

- API key via env var.
- Check header X-API-Key.
- Return 401/403 appropriately.

Live demo (5–10 min)

65. Demo: protect POST/PUT/DELETE; GET remains open.
66. Show how to test with curl header.

Hands-on lab (25–35 min)

Lab: Protect write endpoints

- Read API key from env var.

- Require X-API-Key for POST/PUT/DELETE.

- Return 401/403 with JSON error if missing/wrong.

Completion criteria

- Write endpoints require key.
- Errors are correct and consistent.

Common pitfalls to watch for

- Hard-coding key.
- Applying protection inconsistently across endpoints.

Optional extensions (stay in Advanced scope)

- Add rate-limit-like behavior (toy: sleep after 5 failures) - optional.

Quick check / exit ticket

Quick check: What status code fits ‘missing/invalid credentials’?

Hour 38: Consuming your own API: Python client functions

Outcomes

- Write a small client library using requests.
- Handle errors and timeouts properly.

Instructor talk points (10–20 min)

- Client functions mirror API endpoints.
- Timeouts + raising for status.
- Mapping errors to user messages.

Live demo (5–10 min)

67. Demo: client.create_record() then client.list_records().

Hands-on lab (25–35 min)

Lab: Build a client

- Create client/api_client.py with:

- list_records

- create_record

- update_record

- delete_record

- Add timeout to requests.

- Print friendly messages for common failures.

Completion criteria

- Client functions work end-to-end.
- Timeouts set.
- Errors handled gracefully.

Common pitfalls to watch for

- No timeout; hanging call.
- Not passing API key header for write ops.

Optional extensions (stay in Advanced scope)

- Add a simple CLI that uses the client.

Quick check / exit ticket

Quick check: Why should the client set a timeout by default?

Hour 39: Integration choice: GUI talks to API OR parallel UI/API

Outcomes

- Choose and implement one integration approach.
- Demonstrate a cohesive system.

Instructor talk points (10–20 min)

- Option A: GUI -> API -> service -> DB.
- Option B: GUI uses service+DB locally; API is a separate interface.

Live demo (5–10 min)

68. Demo: one end-to-end flow for chosen option.

Hands-on lab (25–35 min)

Lab: Integration sprint

Choose one:

A) Modify GUI to call the API for CRUD.

B) Keep GUI local and ensure API uses same service layer.

Minimum: demonstrate both surfaces operating on the same underlying data store (DB).

Completion criteria

- Chosen integration path works.
- End-to-end flow demonstrated.

Common pitfalls to watch for

- Two different schemas/fields between GUI and API.
- Forgetting to refresh GUI after API calls.

Optional extensions (stay in Advanced scope)

- Add a status indicator in GUI showing API connectivity (optional).

Quick check / exit ticket

Quick check: What's the biggest tradeoff of your chosen architecture?

Hour 40: Checkpoint 5: API milestone demo

Outcomes

- Deliver a working REST API (and optionally a client) backed by SQLite.

Instructor talk points (10–20 min)

- Grading focus: correctness, contract consistency, safe errors, and structure.

Live demo (5–10 min)

69. Demo: fast-grade checklist: health, list, create, update, delete, auth gate.

Hands-on lab (25–35 min)

Checkpoint 5 (45–60 min)

Deliver:

- Flask API with CRUD

- SQLite persistence

- consistent JSON errors

- API key gate for writes

- (bonus) Python client

Short demo: curl or client script showing operations.

Completion criteria

- API works reliably.
- Persistence proven.
- Security gate works.

Common pitfalls to watch for

- Returning HTML errors.
- Uncaught exceptions causing 500s.
- No separation between route and service logic.

Optional extensions (stay in Advanced scope)

- Add search endpoint with paging.

Quick check / exit ticket

Quick check: What’s one thing you did to keep your API maintainable?

Day 6

Session 11 (Hours 41–44)

Hour 41: Pandas essentials: loading, cleaning, summarizing

Outcomes

- Load data into a DataFrame.
- Compute basic summaries and clean common issues.

Instructor talk points (10–20 min)

- CSV import/export.
- Selecting columns, filtering rows.
- Missing values basics.

Live demo (5–10 min)

70. Demo: load exported CSV from capstone; compute counts and groupby summary.

Hands-on lab (25–35 min)

Lab: Analytics starter

- Export your records to CSV (if not already).

- Load into pandas.

- Compute at least 3 metrics (count, mean/sum, group summary).

- Save a summary table to reports/summary.csv.

Completion criteria

- Data loads successfully.
- Summary metrics computed and saved.

Common pitfalls to watch for

- Wrong delimiter/encoding.
- Mixing numeric strings with numbers.

Optional extensions (stay in Advanced scope)

- Add a date column and compute records per day/week (optional).

Quick check / exit ticket

Quick check: What does DataFrame.head() help you confirm?

Hour 42: Visualization with matplotlib: charts you can ship

Outcomes

- Create at least one readable chart.
- Save plots to files for reports.

Instructor talk points (10–20 min)

- Labels, titles, axis rotation (minimal).
- Saving figures to PNG.

Live demo (5–10 min)

71. Demo: bar chart from groupby counts; save to reports/plot.png.

Hands-on lab (25–35 min)

Lab: Generate charts

- Create 1–2 plots from your dataset (e.g., count by category/status; time trend).

- Save charts into reports/.

- Make sure charts have labels and a title.

Completion criteria

- Charts saved to reports/ and are readable.

Common pitfalls to watch for

- Forgetting plt.close() causing overlapping plots.
- Unlabeled axes.

Optional extensions (stay in Advanced scope)

- Add a second chart type (line or histogram).

Quick check / exit ticket

Quick check: Why should you save plots to files in an automated workflow?

Hour 43: Regression demo (practical, limited scope)

Outcomes

- Run a simple regression workflow (optional if dataset fits).
- Explain train/test split at a high level.

Instructor talk points (10–20 min)

- When regression makes sense.
- Train/test split concept.
- Evaluate with a simple metric (R^2 or MAE).

Live demo (5–10 min)

72. Demo: tiny regression example (synthetic or provided dataset).

Hands-on lab (25–35 min)

Lab: Regression mini-lab (if supported)

- Use a provided dataset or a cleaned dataset with numeric features.

- Run a simple linear regression (scikit-learn if available).

- Report one metric and one observation about model fit.

If regression isn't appropriate, do an additional analysis + chart instead.

Completion criteria

- A regression run OR equivalent analysis completed.
- Learner can explain what the metric means.

Common pitfalls to watch for

- Garbage in/garbage out from uncleaned data.
- Mixing categorical strings into numeric model without encoding (skip encoding in this course).

Optional extensions (stay in Advanced scope)

- Try a second feature and compare performance.

Quick check / exit ticket

Quick check: Why do we separate training data from testing data?

Hour 44: Integrate analytics into capstone (report feature)

Outcomes

- Add a usable analytics/export feature to the capstone.
- Generate a repeatable report.

Instructor talk points (10–20 min)

- Report pipeline: export -> analyze -> chart -> save artifacts.
- Keep it runnable with one command if possible.

Live demo (5–10 min)

73. Demo: python -m reports.generate_report produces summary + plots.

Hands-on lab (25–35 min)

Lab: Report integration

- Add a command or button that generates:

- CSV export

- summary table

- at least one plot saved to reports/

- Document how to run it in README.

Completion criteria

- One-command report generation works.
- Artifacts are saved and documented.

Common pitfalls to watch for

- Hard-coded paths that break on another machine.
- Report depends on manual steps.

Optional extensions (stay in Advanced scope)

- Add a 'report date' stamp to file names.

Quick check / exit ticket

Quick check: What makes a report feature ‘repeatable’?

Session 12 (Hours 45–48)

Hour 45: Testing with pytest: unit tests for service layer

Outcomes

- Write unit tests for core logic.
- Use fixtures for shared setup.

Instructor talk points (10–20 min)

- Test pyramid (pragmatic).
- Arrange/Act/Assert.
- Fixtures for sample records and temp DB.

Live demo (5–10 min)

74. Demo: test_validation_error, test_add_and_get, using tmp_path fixture.

Hands-on lab (25–35 min)

Lab: Add tests

- Create at least 5 tests for service/repo logic.

- Include 2 negative tests (invalid input, missing record).

- Run pytest and confirm all pass.

Completion criteria

- Tests run and pass.
- Negative tests assert correct exceptions.

Common pitfalls to watch for

- Testing UI directly (skip).
- Brittle tests that depend on record ordering.

Optional extensions (stay in Advanced scope)

- Add a fixture that creates an isolated SQLite DB per test run.

Quick check / exit ticket

Quick check: What should a good unit test avoid (external dependencies, randomness, time)?

Hour 46: Coverage + edge cases + lightweight integration test

Outcomes

- Measure coverage and improve the most important gaps.
- Add one integration-style test.

Instructor talk points (10–20 min)

- Coverage basics (what it is, what it isn't).
- Edge cases from bug history.
- One integration test: service+repo together.

Live demo (5–10 min)

75. Demo: run coverage report.
76. Add a test that hits SQLite repo end-to-end.

Hands-on lab (25–35 min)

Lab: Harden with tests

- Run coverage (or a simple report tool if available).

- Add 2 tests to cover the highest-value missing branches.

- Add 1 integration test (repo + service together).

Completion criteria

- Coverage improved (target is 'reasonable', not perfect).
- Integration test passes.

Common pitfalls to watch for

- Chasing 100% coverage at the expense of meaningful tests.
- Testing Flask routes heavily (keep minimal in this course).

Optional extensions (stay in Advanced scope)

- Add a test for API key protection logic (pure function) if structured for it.

Quick check / exit ticket

Quick check: Why can high coverage still miss bugs?

Hour 47: Packaging and delivery: requirements + runnable app + optional executable

Outcomes

- Create a reproducible run setup.
- Optionally build an executable for the GUI or API.

Instructor talk points (10–20 min)

- venv creation, requirements.txt, pip install -r.
- Entry points: python -m ...
- Optional: PyInstaller overview.

Live demo (5–10 min)

77. Demo: create venv, install requirements, run app from clean environment.
78. Optional: pyinstaller --onefile for GUI.

Hands-on lab (25–35 min)

Lab: Ship it

- Create requirements.txt (pip freeze or curated list).

- Write clear run instructions in README.

- Demonstrate running from a clean venv.

- Optional: build an executable (GUI) or a packaged run script (API).

Completion criteria

- Project runs from fresh environment.
- README has clear steps.
- Optional deliverable produced if time.

Common pitfalls to watch for

- OS-specific paths in README.
- Missing dependencies.
- Executable builds failing due to hidden imports.

Optional extensions (stay in Advanced scope)

- Add a 'make_report' command or script.

Quick check / exit ticket

Quick check: If a teammate cloned your repo, what are the exact 3 commands they'd run to use it?

Hour 48: Final: capstone demo + certification-style review + retrospective

Outcomes

- Demonstrate an end-to-end working system.
- Review PCAP-style concepts and code-reading skills.

Instructor talk points (10–20 min)

- Demo rubric and timeboxing.
- PCAP-style review: OOP, exceptions, modules, files, data handling, code comprehension.
- Retrospective: what to practice next.

Live demo (5–10 min)

79. Demo: code-reading question and predict output.
80. Show how you would answer an exam-style 'what prints' question.

Hands-on lab (25–35 min)

Final (60–90 min)

1) Capstone demo (3–5 minutes each or groups)

2) Short certification-style quiz (15–20 min)

3) Individual feedback and next-step plan

Capstone must show: model/service, persistence, one UI surface (GUI or API), report, tests, packaging readiness.

Completion criteria

- Capstone meets minimum deliverables.
- Learner can explain architecture choices.

Common pitfalls to watch for

- Overrunning demo time.
- Last-minute refactor breaks working project.

Optional extensions (stay in Advanced scope)

- Stretch: demo both GUI and API operating over same DB.

Quick check / exit ticket

Quick check: Name one skill you'll apply immediately at work, and one you'll keep practicing.

Checkpoint & capstone grading rubrics

Checkpoint rubric (use for Checkpoints 1–5)

Score each category 0–3. Suggested pass threshold: 9/15 (adjust as needed).

Capstone rubric (final)

Troubleshooting quick guide (Advanced)

Out-of-scope for this Advanced course (avoid rabbit holes)

Full authentication/authorization systems (OAuth, JWT), production security engineering

Asyncio/concurrency deep dives, microservices architecture

Deep learning or complex ML pipelines, feature engineering, hyperparameter tuning

Production-grade CI/CD, containers/Kubernetes, cloud deployments

Full ORM mastery (migrations frameworks, advanced session management)

Checkpoint & capstone grading rubrics

Checkpoint rubric (use for Checkpoints 1–5)

Score each category 0–3. Suggested pass threshold: 10/15 (adjust to your program requirements).

Category	What you're looking for
Correctness	Meets functional requirements; workflows behave correctly for typical test cases.
Architecture & clarity	Clear separation (core vs UI/API), readable design, minimal duplication.
Error handling	Custom exceptions used; failures handled cleanly; friendly messages where appropriate.
Persistence / integration	Save/load or DB operations work reliably and are demonstrated.
Professional habits	Uses Git commits, sensible naming, logs/tests where required in that checkpoint.

Capstone rubric (final)

Category	What you're looking for
Minimum deliverables	Core + SQLite + one UI surface (GUI or API) + analytics + tests + packaging instructions.
End-to-end reliability	CRUD flows work, persistence proven across restarts, and common failures handled.
UI/API quality	Usable UI flow or clean REST contracts; validation and errors are consistent.
Analytics	Export + at least one chart saved and explained; optional regression demo if included.
Testing & quality	Meaningful pytest coverage of core logic; edge cases tested; coverage report produced (basic).
Demo & explanation	Clear demo in <5 minutes; learner can explain design decisions and tradeoffs.

Troubleshooting quick guide (Advanced)

Symptom	Fast fix
ModuleNotFoundError / import issues	Confirm you're running from the project root; prefer 'python -m <package>' for packages; avoid naming files after stdlib modules.
pip install fails	Check whether installs are allowed; confirm correct venv; try upgrading pip; use cached wheels if policy requires.
GUI freezes	Don't run long tasks on the UI thread; keep callbacks short; add status updates; avoid blocking network calls in callbacks.
SQLite changes not saved	Ensure commit() is called; use context managers; confirm DB path is what you expect.
UPDATE/DELETE affects wrong row	Use stable IDs; verify WHERE clause; print the ID being used before executing.
Flask route not hit	Confirm method (GET/POST), URL, and app is running on expected port; check debug logs.
API returns 500	Capture traceback in logs, return consistent error payloads, validate request JSON before using it.
pandas dtype surprises	Inspect df.dtypes; convert with to_numeric/to_datetime; handle missing values explicitly.
pytest finds no tests	Ensure files are named test_*.py and functions start with test_; run from project root.
Coverage missing core lines	Run coverage against the same command you use to run tests; avoid excluding src/ unintentionally.

Out-of-scope stretch topics (keep optional)

- Asyncio and concurrent programming patterns
- Docker/containerization and cloud deployment
- Advanced security/auth (OAuth/JWT), full RBAC systems
- Complex ML workflows beyond a small regression demo