Instructor Runbook
Python Programming (Basic) – 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 a teaching companion for a 48-hour Python Basics course delivered as 12 sessions of 4 instructional hours each. It includes: session flow, hour-by-hour lesson blocks, live demo steps, lab prompts, checkpoint assessments, rubrics, and troubleshooting tips.

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.

Course outcomes (Basic)

- Set up a Python 3 development environment and run scripts.
- Use core data types (numbers, strings, booleans) and operators.
- Work with core data structures (lists, tuples, sets, dictionaries).
- Implement control flow using conditionals and loops.
- Write and organize code with functions and modules; introduce basic classes.
- Read/write files (text and JSON) and handle common errors with exceptions.
- Build and present a small command-line capstone project.

Pre-course instructor checklist

- Confirm lab access and credentials for all learners; test logins from a non-instructor account.
- Verify Python 3 is available in the lab image/VM; confirm VS Code (or editor) launches cleanly.
- Open a sample project folder and run a simple script to confirm permissions and file paths work.
- Confirm internet access policy (if any) and whether installing packages is allowed (Basics can be done without pip).
- Prepare a shared class repo or zip of starter files (blank templates + checkpoint starter code).
- Decide how you will collect labs (screenshare, LMS upload, Git, or file submission).
- Post the ‘common errors’ quick sheet (indentation, type conversion, file paths).

Session delivery rhythm (recommended)

- Start-of-session (5–10 min): recap + 1 warm-up question.
- Teach (10–15 min): concept + examples + quick questions.
- Demo (5–10 min): live coding with narration; ask learners to predict output.
- Lab (25–35 min): guided → independent; circulate and spot-check.
- Debrief (5 min): 2–3 learners share solutions; highlight common fixes.

48-hour hour-by-hour lesson blocks

Each hour block includes: outcomes, instructor talk points, demo, lab prompt, completion criteria, common pitfalls, and optional extensions (still within Basics scope).

Session overviews (quick schedule)

Session 1 overview (Hours 1–4)

Hour	Focus
1	Orientation + environment readiness
2	First scripts: print(), comments, and reading errors
3	Variables + basic types
4	Numbers + operators

Session 2 overview (Hours 5–8)

Hour	Focus
5	Strings fundamentals: indexing, slicing, len()
6	String methods: normalize, search, replace
7	Input/output + type conversion
8	Checkpoint 1: Fundamentals mini-assessment

Session 3 overview (Hours 9–12)

Hour	Focus
9	Comparisons + boolean logic
10	String formatting with f-strings
11	Working with text: split/join
12	Debugging habits (Basics level)

Session 4 overview (Hours 13–16)

Hour	Focus
13	Lists fundamentals
14	Iterating lists with for-loops (gentle intro)
15	Nested lists (table-like data)
16	Checkpoint 2: Lists + string handling

Session 5 overview (Hours 17–20)

Hour	Focus
17	Tuples + unpacking
18	Sets: uniqueness + membership
19	Dictionaries fundamentals

20	Dictionary iteration + counting pattern

Session 6 overview (Hours 21–24)

Hour	Focus
21	Choosing the right structure
22	Data-structure drill circuit (guided practice)
23	Mini-project: In-memory tracker
24	Checkpoint 3: Data structures assessment

Session 7 overview (Hours 25–28)

Hour	Focus
25	Conditionals: if/elif/else and boundaries
26	while loops + sentinel patterns
27	for loops + range()
28	Loop patterns: counters, accumulators, min/max

Session 8 overview (Hours 29–32)

Hour	Focus
29	Menu loops (CLI pattern)
30	Input validation (Basics approach)
31	Mini-project: CLI Contact Manager (in-memory)
32	Checkpoint 4: Control flow assessment

Session 9 overview (Hours 33–36)

Hour	Focus
33	Functions: def, parameters, return
34	Scope + common function mistakes
35	Functions with collections
36	Modules: imports and creating utils.py

Session 10 overview (Hours 37–40)

Hour	Focus
37	Intro classes: objects, attributes, methods
38	Collections of objects + searching
39	Basic encapsulation + data validation
40	Checkpoint 5: Functions + modules + intro OOP

Session 11 overview (Hours 41–44)

Hour	Focus
41	Files: reading and writing text
42	JSON persistence (stdlib json)
43	Directories + paths (pathlib preferred)
44	Exception handling (full Basics treatment)

Session 12 overview (Hours 45–48)

Hour	Focus
45	Capstone kickoff: requirements + scaffolding
46	Capstone build sprint
47	Capstone polish + demo readiness
48	Final assessment + certification-style review

Session 1 (Hours 1–4)

Hour 1: Orientation + environment readiness

Outcomes

- Access the lab/VM and run Python.
- Create a workspace folder and run a first script.

Instructor talk points (10–15 min)

- Course workflow: lecture → demo → lab.
- How Python executes: interpreter vs script file.
- Where files live in the lab; naming and folders.

Live demo (5–10 min)

1. Open terminal / IDE.
2. Run: python --version
3. Create folder: python_basics/
4. Create hello_course.py and run it.

Hands-on lab (25–35 min)

Lab: Environment check + first run

- Create a folder for the course.

- Create hello_course.py that prints your name and today's date.

- Run it from the terminal and from the IDE (if available).

Completion criteria

- Script runs without errors from both terminal and IDE.
- Learner can find the output and locate the file on disk.

Common pitfalls to watch for

- Wrong interpreter selected in IDE.
- Saving file outside workspace.
- Confusing terminal path vs file path.

Optional extensions (stay in Basics scope)

- Add a second script that imports datetime and prints formatted date.
- Add a 'run instructions' comment header.

Quick check / exit ticket

Quick check: What's the difference between running in the REPL and running a .py file?

Hour 2: First scripts: print(), comments, and reading errors

Outcomes

- Use print() to produce output.
- Explain indentation and syntax rules at a basic level.

Instructor talk points (10–15 min)

- print() and string literals.
- Comments (#) and why we comment.
- Common beginner errors: missing quotes, parentheses, typos.

Live demo (5–10 min)

5. Live-code 5 common mistakes and read the error message out loud.
6. Show how to fix each in <30 seconds.

Hands-on lab (25–35 min)

Lab: Greeter

- Write a program that prints a 3-line greeting.

- Add a comment describing what the program does.

- Intentionally create a syntax error, read the message, then fix it.

Completion criteria

- Output prints 3 lines correctly.
- Learner can describe what changed when fixing the error.

Common pitfalls to watch for

- Using smart quotes from copy/paste.
- Missing closing parentheses/quotes.

Optional extensions (stay in Basics scope)

- Print a simple ASCII box around the greeting using \n and spacing.

Quick check / exit ticket

Quick check: Why is whitespace/indentation important in Python?

Hour 3: Variables + basic types

Outcomes

- Create variables and assign values.
- Use type() to inspect values.

Instructor talk points (10–15 min)

- Identifiers and naming rules.
- Strings vs numbers vs booleans.
- Reassignment and reading from right to left.

Live demo (5–10 min)

7. Demo: build a small ‘profile card’ with variables; change values and re-run.

Hands-on lab (25–35 min)

Lab: Profile Card

- Store name, city, favorite number, and a boolean (e.g., likes_python).

- Print a formatted ‘card’ showing values.

- Use type() for each value and print the result.

Completion criteria

- Program prints values and types correctly.
- Learner uses clear variable names.

Common pitfalls to watch for

- Overwriting variable with a different type accidentally.
- NameError from typos.

Optional extensions (stay in Basics scope)

- Add a computed value (favorite number + 10).
- Add user input for one field (keep it simple).

Quick check / exit ticket

Quick check: What’s the difference between '5' and 5?

Hour 4: Numbers + operators

Outcomes

- Use arithmetic operators and precedence.
- Explain integer vs float results.

Instructor talk points (10–15 min)

- + - * / // % ** basics.
- Order of operations.
- round() for display, not for storage.

Live demo (5–10 min)

8. Demo: mini calculator; show / vs //; show % with even/odd check.

Hands-on lab (25–35 min)

Lab: Tip Calculator

- Ask for bill total and tip percent.

- Compute tip and final total.

- Display to 2 decimals using round() or formatting.

Completion criteria

- Correct totals for sample inputs.
- Uses float conversion where needed.

Common pitfalls to watch for

- Forgetting to convert input() to float.
- Integer division surprises.
- Rounding too early.

Optional extensions (stay in Basics scope)

- Add split count (per person).
- Add sales tax as an extra percent.

Quick check / exit ticket

Quick check: What does % return and when is it useful?

Session 2 (Hours 5–8)

Hour 5: Strings fundamentals: indexing, slicing, len()

Outcomes

- Index and slice strings.
- Use len() and basic membership checks.

Instructor talk points (10–15 min)

- String indexing starts at 0.
- Slicing basics: s[a:b].
- len() and bounds.

Live demo (5–10 min)

9. Demo: extract initials, domain from email, and last character with -1.

Hands-on lab (25–35 min)

Lab: Username Builder

- Input first and last name.

- Build a username: first initial + last name (lowercase).

- Show username length.

Completion criteria

- Correct username for multi-word last names (strip spaces).
- Handles at least one realistic name.

Common pitfalls to watch for

- IndexError from empty strings.
- Not normalizing case/spacing.

Optional extensions (stay in Basics scope)

- Add a rule: if username > 12 chars, shorten last name.
- Add a numeric suffix if desired.

Quick check / exit ticket

Quick check: What does s[-1] do?

Hour 6: String methods: normalize, search, replace

Outcomes

- Use common string methods.
- Explain immutability: methods return new strings.

Instructor talk points (10–15 min)

- .lower(), .upper(), .strip(), .replace(), .find()
- Using 'in' to check containment.

Live demo (5–10 min)

10. Demo: clean messy input; show before/after; show find() returning -1.

Hands-on lab (25–35 min)

Lab: Text Sanitizer

- Input a sentence.

- Strip whitespace, make lowercase.

- Replace multiple spaces with single spaces (basic approach).

- Report whether a chosen keyword appears.

Completion criteria

- Outputs transformed sentence.
- Correctly reports keyword presence.

Common pitfalls to watch for

- Expecting replace to modify in place.
- Confusing find() results.

Optional extensions (stay in Basics scope)

- Add: count occurrences using .count().
- Add: title-case name fields.

Quick check / exit ticket

Quick check: Why do we assign back after calling a string method?

Hour 7: Input/output + type conversion

Outcomes

- Collect user input and convert types safely (happy path).
- Explain why input() returns a string.

Instructor talk points (10–15 min)

- input(), int(), float()
- Simple validation idea: re-prompt later (preview only).

Live demo (5–10 min)

11. Demo: small converter; show ValueError if input is not numeric (save full handling for exceptions day).

Hands-on lab (25–35 min)

Lab: Unit Converter

- Choose: miles→km OR F→C.

- Ask user for a number.

- Convert and print formatted output.

Completion criteria

- Correct conversion on sample input.
- Readable output with units.

Common pitfalls to watch for

- ValueError when user types text.
- Hard-coded constants wrong.

Optional extensions (stay in Basics scope)

- Add a menu to choose conversion type (no loops yet).

Quick check / exit ticket

Quick check: What happens if you call int() on '3.5'?

Hour 8: Checkpoint 1: Fundamentals mini-assessment

Outcomes

- Demonstrate basic script writing, variables, numbers, strings, and input.

Instructor talk points (10–15 min)

- Explain rules: open-book, individual, timeboxed.
- Remind: focus on correctness first, then formatting.

Live demo (5–10 min)

12. Demo: walk through sample rubric; show how you'll grade.

Hands-on lab (25–35 min)

Checkpoint Lab 1 (45–60 min)

Build: Simple Receipt Generator

- Input: item name, quantity, price per item.

- Compute subtotal and total.

- Print a receipt with aligned lines and 2-decimal money formatting.

Optional quiz (10 min): 8–10 quick questions (types, operators, strings).

Completion criteria

- Program runs end-to-end.
- Correct math and conversions.
- Output readable.

Common pitfalls to watch for

- Forgetting float conversion.
- String concatenation vs numeric addition.

Optional extensions (stay in Basics scope)

- Add: discount percent.
- Add: tax percent.

Quick check / exit ticket

Quick check: Ask 2 learners to explain how they debugged one error.

Day 2

Session 3 (Hours 9–12)

Hour 9: Comparisons + boolean logic

Outcomes

- Use comparison operators.
- Combine conditions with and/or/not.

Instructor talk points (10–15 min)

- == vs =.
- Chaining comparisons (keep simple).
- Truthy/falsey preview (keep minimal).

Live demo (5–10 min)

13. Demo: age gating; show two comparisons combined with and.

Hands-on lab (25–35 min)

Lab: Eligibility Checker

- Input age.

- Determine eligibility for 3 brackets (child/teen/adult) using comparisons.

- Print a clear message.

Completion criteria

- Correct branching for boundary values.
- Uses >= and < appropriately.

Common pitfalls to watch for

- Off-by-one boundaries.
- Comparing strings to numbers.

Optional extensions (stay in Basics scope)

- Add a second condition (e.g., has_membership).

Quick check / exit ticket

Quick check: What’s the difference between == and = ?

Hour 10: String formatting with f-strings

Outcomes

- Use f-strings for readable output.
- Format numbers to fixed decimals.

Instructor talk points (10–15 min)

- f"{var}" basics.
- Formatting: {value:.2f}
- Why formatting is better than manual rounding + concatenation.

Live demo (5–10 min)

14. Demo: take Tip Calculator and reformat output; show alignment with format spec.

Hands-on lab (25–35 min)

Lab: Upgrade Tip Calculator

- Rewrite output using f-strings.

- Show bill, tip, total each on a separate labeled line.

- Format money to 2 decimals.

Completion criteria

- Clean, consistent output formatting.
- No type errors.

Common pitfalls to watch for

- Forgetting the leading f.
- Using braces incorrectly.

Optional extensions (stay in Basics scope)

- Align columns using width specifiers (e.g., {label:<10}).

Quick check / exit ticket

Quick check: Why is f-string formatting usually better than + concatenation?

Hour 11: Working with text: split/join

Outcomes

- Split sentences into lists of words.
- Join words back into a string.

Instructor talk points (10–15 min)

- .split() and default whitespace behavior.
- join() pattern: ' '.join(list).
- Counting words.

Live demo (5–10 min)

15. Demo: compute word count and rebuild sentence with hyphens.

Hands-on lab (25–35 min)

Lab: Sentence Stats

- Input a sentence.

- Print word count.

- Print longest word.

- Print a version where words are joined by ' | '.

Completion criteria

- Accurate count and longest word.
- Uses split/join correctly.

Common pitfalls to watch for

- Punctuation attached to words.
- Empty input edge case.

Optional extensions (stay in Basics scope)

- Strip punctuation for bonus (basic: replace(',', '') etc.).

Quick check / exit ticket

Quick check: What type does split() return?

Hour 12: Debugging habits (Basics level)

Outcomes

- Interpret common error messages.

- Use print-debugging strategically.

Instructor talk points (10–15 min)

- Reading tracebacks top-to-bottom.
- Common errors: NameError, TypeError, ValueError, IndexError.
- Using small test inputs.

Live demo (5–10 min)

16. Demo: purposely break a script; add print statements to isolate where it fails.

Hands-on lab (25–35 min)

Lab: Debugging Drill

- You'll receive 3 broken scripts.

- For each: run, read error, hypothesize cause, fix.

- Write a 1-sentence explanation of what went wrong.

Completion criteria

- All scripts fixed.
- Learner can explain each fix.

Common pitfalls to watch for

- Changing multiple things at once.
- Not re-running after each small fix.

Optional extensions (stay in Basics scope)

- Add a 'sanity print' of variable types before the crash.

Quick check / exit ticket

Quick check: When is it helpful to print type(variable)?

Session 4 (Hours 13–16)

Hour 13: Lists fundamentals

Outcomes

- Create and modify lists.
- Use append/pop/remove and membership tests.

Instructor talk points (10–15 min)

- List literals.
- Indexing and slicing lists.
- Mutability vs strings.

Live demo (5–10 min)

17. Demo: build shopping list; append, remove, pop; print after each operation.

Hands-on lab (25–35 min)

Lab: Shopping List

- Start with an empty list.

- Ask user for 5 items (one at a time) and append.

- Remove one chosen item.

- Print the final list and item count.

Completion criteria

- List updates correctly.
- Handles removing an item that exists.

Common pitfalls to watch for

- remove() on missing item raises error (note it; fix later with condition or try/except).
- Confusing pop index vs value.

Optional extensions (stay in Basics scope)

- Allow ‘done’ to stop early.
- Sort list alphabetically (preview: sorted()).

Quick check / exit ticket

Quick check: Why can lists change but strings cannot?

Hour 14: Iterating lists with for-loops (gentle intro)

Outcomes

- Use for-loops to process list items.
- Use accumulators for totals/averages.

Instructor talk points (10–15 min)

- for item in items.
- Accumulator pattern: total += x.
- Counting items that meet a condition.

Live demo (5–10 min)

18. Demo: sum numbers in a list; compute average.

Hands-on lab (25–35 min)

Lab: Grade Average

- Ask for 5 numeric grades.

- Store in a list.

- Compute average and highest grade.

- Print results with formatting.

Completion criteria

- Correct average and max.
- Uses a loop or built-in max().

Common pitfalls to watch for

- Not converting input to float/int.
- Dividing by wrong count.

Optional extensions (stay in Basics scope)

- Drop the lowest grade (basic: remove min).

Quick check / exit ticket

Quick check: What does ‘for x in my_list’ give you each iteration?

Hour 15: Nested lists (table-like data)

Outcomes

- Represent simple 2D data.
- Access elements with row/column indices.

Instructor talk points (10–15 min)

- List of lists concept.
- Iterating rows.
- Keep examples small and concrete.

Live demo (5–10 min)

19. Demo: seating chart; print rows; update one seat.

Hands-on lab (25–35 min)

Lab: Seating Chart

- Create a 3x3 seating chart (names or seat IDs).

- Print it as rows.

- Ask for row/col and change that seat to 'X'.

- Reprint the chart.

Completion criteria

- Correct access and update.
- Reprints properly.

Common pitfalls to watch for

- Mixing up row/col indexing.
- IndexError due to out-of-range input.

Optional extensions (stay in Basics scope)

- Validate row/col are 0–2 (basic if-check).

Quick check / exit ticket

Quick check: How do you access the middle item of a 3x3 list-of-lists?

Hour 16: Checkpoint 2: Lists + string handling

Outcomes

- Demonstrate list creation, iteration, and formatted output.

Instructor talk points (10–15 min)

- Explain grading focus: correct list use + clean output.

Live demo (5–10 min)

20. Demo: sample solution walkthrough at a high level (don’t give full code).

Hands-on lab (25–35 min)

Checkpoint Lab 2 (45–60 min)

Build: Simple To-Do List (in-memory)

- Menu options (no loop required yet): add item, remove item, show items.

- Store items in a list.

- Show numbered list output.

Optional quiz (10 min): strings + lists + loops basics.

Completion criteria

- Can add and remove items.
- Displays numbered output.
- No crashes on typical inputs.

Common pitfalls to watch for

- Remove by name vs remove by index confusion.
- Not reprinting after change.

Optional extensions (stay in Basics scope)

- Add a priority tag (e.g., '[H]') as part of the string.
- Allow filtering by keyword using 'in'.

Quick check / exit ticket

Quick check: What’s the difference between remove() and pop()?

Day 3

Session 5 (Hours 17–20)

Hour 17: Tuples + unpacking

Outcomes

- Create tuples and explain immutability.
- Unpack tuple values into variables.

Instructor talk points (10–15 min)

- Tuple literals and parentheses.
- When tuples are useful (fixed-size records).
- Unpacking pattern.

Live demo (5–10 min)

21. Demo: return-like tuple (x, y); unpack; show immutability error.

Hands-on lab (25–35 min)

Lab: Coordinate Tracker

- Store 5 (x, y) points as tuples in a list.

- Print each point.

- Compute min and max x values.

Completion criteria

- Uses tuples correctly.
- Correct min/max results.

Common pitfalls to watch for

- Confusing (x) with (x,) single-item tuple.
- Trying to modify tuple item.

Optional extensions (stay in Basics scope)

- Compute distance from origin (basic math).

Quick check / exit ticket

Quick check: Why might you choose a tuple over a list?

Hour 18: Sets: uniqueness + membership

Outcomes

- Use a set to remove duplicates.
- Perform membership checks efficiently.

Instructor talk points (10–15 min)

- Set literals, add(), and automatic uniqueness.
- Common set operations (union/intersection) at a conceptual level.

Live demo (5–10 min)

22. Demo: list of names → set; show duplicates removed.

Hands-on lab (25–35 min)

Lab: Unique Visitors

- Input 10 names (allow repeats).

- Store in a list.

- Convert to a set and print unique count and names.

Completion criteria

- Correct unique count.
- Explains why duplicates disappear.

Common pitfalls to watch for

- Expecting order preserved (sets are unordered).

Optional extensions (stay in Basics scope)

- Show both: sorted(unique_set) for display.

Quick check / exit ticket

Quick check: When is a set better than a list?

Hour 19: Dictionaries fundamentals

Outcomes

- Create dicts and use keys/values.
- Safely access with get().

Instructor talk points (10–15 min)

- Key/value idea.
- Adding/updating entries.
- get() with default.

Live demo (5–10 min)

23. Demo: inventory dict; update quantities; show KeyError vs get().

Hands-on lab (25–35 min)

Lab: Inventory

- Create a dict with 5 items and quantities.

- Ask user for an item name.

- If it exists, increase quantity by a number.

- Print inventory neatly.

Completion criteria

- Updates correct key.
- Does not crash on missing key (use get or in-check).

Common pitfalls to watch for

- Case sensitivity issues.
- Accidental new key due to spelling mismatch.

Optional extensions (stay in Basics scope)

- Normalize keys to lowercase.
- Print sorted by item name.

Quick check / exit ticket

Quick check: What happens if you access a missing key with dict[key]?

Hour 20: Dictionary iteration + counting pattern

Outcomes

- Iterate through dict items.
- Build a frequency counter.

Instructor talk points (10–15 min)

- items() loop.
- Counting pattern: d[word] = d.get(word, 0) + 1.

Live demo (5–10 min)

24. Demo: word frequency from a sentence.

Hands-on lab (25–35 min)

Lab: Word Counter

- Input a sentence.

- Split into words.

- Build a frequency dictionary.

- Print each word and count.

Completion criteria

- Correct counts.
- Uses get() pattern or if-check.

Common pitfalls to watch for

- Punctuation included.
- Case inconsistencies.

Optional extensions (stay in Basics scope)

- Normalize to lowercase and strip punctuation (simple replace).

Quick check / exit ticket

Quick check: What does dict.items() produce?

Session 6 (Hours 21–24)

Hour 21: Choosing the right structure

Outcomes

- Select list vs set vs dict based on task.
- Refactor an approach using a better structure.

Instructor talk points (10–15 min)

- Examples: ordered collection vs uniqueness vs lookup.
- Tradeoffs: readability and simplicity.

Live demo (5–10 min)

25. Demo: membership lookup in list vs set (conceptual; no benchmarking needed).

Hands-on lab (25–35 min)

Lab: Refactor Challenge

- Given a small problem (e.g., checking duplicates or counting), first solve using a list.

- Then refactor to set/dict and explain why it's better.

Completion criteria

- Working refactor.
- Learner explanation includes at least one tradeoff.

Common pitfalls to watch for

- Overengineering (using dict when list is simplest).

Optional extensions (stay in Basics scope)

- Add a small feature: search/filter results.

Quick check / exit ticket

Quick check: If you need fast lookup by name, which structure fits best?

Hour 22: Data-structure drill circuit (guided practice)

Outcomes

- Practice with lists/tuples/sets/dicts in short timed tasks.

Instructor talk points (10–15 min)

- Explain station rotation and goals.
- Emphasize finishing one task before optimizing.

Live demo (5–10 min)

26. Demo: one station solution quickly; show expected output.

Hands-on lab (25–35 min)

Lab Circuit (4 x 12 min)

1) Filter numbers > 10 (list)

2) Unique emails (set)

3) Word frequency (dict)

4) Coordinates unpacking (tuple)

End with 5 min share-out.

Completion criteria

- Completed at least 3 of 4 stations.
- Explained one solution verbally.

Common pitfalls to watch for

- Rushing and creating syntax errors.
- Not testing with small examples.

Optional extensions (stay in Basics scope)

- Add extra station: nested list table formatting.

Quick check / exit ticket

Quick check: Ask learners to name one thing they'll use dicts for in real work.

Hour 23: Mini-project: In-memory tracker

Outcomes

- Combine data structures into a coherent mini app.
- Practice clean output and updates.

Instructor talk points (10–15 min)

- Define requirements for a small tracker.
- Choose a storage model (list of dicts OR dict of dicts).

Live demo (5–10 min)

27. Demo: scaffold a menu and one action (Add).

Hands-on lab (25–35 min)

Lab: Mini-Project – Tracker (in-memory)

Choose one:

A) Expense tracker

B) Contact list

C) Book library

Minimum features: add, list, search.

Store records using dicts + lists.

No file I/O yet.

Completion criteria

- Meets minimum features.
- Uses at least one dict and one list meaningfully.

Common pitfalls to watch for

- Messy global variables.
- Hard-to-read printing.

Optional extensions (stay in Basics scope)

- Add update/delete feature.
- Sort results before display.

Quick check / exit ticket

Quick check: What data structure did you choose and why?

Hour 24: Checkpoint 3: Data structures assessment

Outcomes

- Demonstrate confident use of core data structures and iteration.

Instructor talk points (10–15 min)

- Explain assessment rules and rubric.
- Remind: handle missing keys safely.

Live demo (5–10 min)

28. Demo: show how to test with 2–3 example inputs.

Hands-on lab (25–35 min)

Checkpoint Lab 3 (45–60 min)

Build: Simple Contacts (in-memory)

- Store contacts with name → phone using a dict.

- Add at least 5 contacts.

- Search by name; print phone if found.

- List all contacts (sorted optional).

Optional quiz (10 min): dict/set/tuple/list questions.

Completion criteria

- Search works for existing and missing names.
- Clear output.
- No KeyError.

Common pitfalls to watch for

- Forgetting to normalize case.
- Printing dict directly without formatting.

Optional extensions (stay in Basics scope)

- Allow partial match search (keyword in name).

Quick check / exit ticket

Quick check: What’s the difference between dict.get() and dict[key]?

Day 4

Session 7 (Hours 25–28)

Hour 25: Conditionals: if/elif/else and boundaries

Outcomes

- Write clear conditional logic.
- Handle boundary values correctly.

Instructor talk points (10–15 min)

- Condition ordering.
- Nested vs flat if/elif.
- Readable boolean expressions.

Live demo (5–10 min)

29. Demo: shipping tiers; show correct boundary handling.

Hands-on lab (25–35 min)

Lab: Shipping Calculator

- Input package weight.

- Compute cost using tiered rules.

- Print cost and category.

Completion criteria

- Correct cost for boundary weights.
- Readable branching.

Common pitfalls to watch for

- Overlapping conditions.
- Wrong order of elif branches.

Optional extensions (stay in Basics scope)

- Add: free shipping threshold.
- Add: fragile surcharge boolean input.

Quick check / exit ticket

Quick check: Why does the order of elif statements matter?

Hour 26: while loops + sentinel patterns

Outcomes

- Use while loops for repeated prompts.
- Use break/continue appropriately.

Instructor talk points (10–15 min)

- Sentinel value patterns (e.g., 'q' to quit).
- Common infinite loop causes.

Live demo (5–10 min)

30. Demo: password attempts loop; show break on success.

Hands-on lab (25–35 min)

Lab: Password Prompt

- Ask for password up to 3 times.

- If correct, print success and stop.

- If not, print locked out.

Completion criteria

- Stops correctly on success.
- Locks after 3 failures.

Common pitfalls to watch for

- Not updating attempt counter.
- Using = instead of == in condition.

Optional extensions (stay in Basics scope)

- Add: show remaining attempts.
- Add: require minimum password length (simple check).

Quick check / exit ticket

Quick check: What variable must change in a while loop to avoid infinite loops?

Hour 27: for loops + range()

Outcomes

- Use for loops with range().
- Explain inclusive/exclusive end in range.

Instructor talk points (10–15 min)

- range(n), range(a,b), range(a,b,step).
- Looping over sequences vs numbers.

Live demo (5–10 min)

31. Demo: multiplication table rows using nested loops (keep small).

Hands-on lab (25–35 min)

Lab: Multiplication Table

- Ask for a number n.

- Print a 1..10 multiplication table for n.

- Use range().

Completion criteria

- Correct outputs 1..10.
- Formatting is readable.

Common pitfalls to watch for

- Off-by-one in range end.
- Not converting input to int.

Optional extensions (stay in Basics scope)

- Print full 1..n grid if time (optional).

Quick check / exit ticket

Quick check: What does range(1, 4) produce?

Hour 28: Loop patterns: counters, accumulators, min/max

Outcomes

- Apply common loop patterns reliably.
- Track min/max and totals.

Instructor talk points (10–15 min)

- Counter pattern.
- Accumulator pattern.
- Initializing min/max safely.

Live demo (5–10 min)

32. Demo: read 5 numbers; compute sum, min, max.

Hands-on lab (25–35 min)

Lab: Number Stats

- Ask user for 5 numbers.

- Compute min, max, sum, average.

- Print results.

Completion criteria

- Correct stats.
- Clean output.

Common pitfalls to watch for

- Initializing min/max to 0 incorrectly.
- Dividing by wrong count.

Optional extensions (stay in Basics scope)

- Allow variable count until 'done' sentinel.

Quick check / exit ticket

Quick check: What’s a safe way to initialize min before looping?

Session 8 (Hours 29–32)

Hour 29: Menu loops (CLI pattern)

Outcomes

- Build a menu-driven while loop.
- Route actions via if/elif.

Instructor talk points (10–15 min)

- Menu display and input.
- Validating choices (basic).
- Separating logic into functions (preview).

Live demo (5–10 min)

33. Demo: menu scaffold with 3 options and a quit option.

Hands-on lab (25–35 min)

Lab: Upgrade Menu

- Turn a previous script into a looped menu.

- Include: add, list, search actions for your chosen data set.

- Keep state in memory.

Completion criteria

- Menu repeats until quit.
- Each action works.

Common pitfalls to watch for

- Forgetting to reprint menu.
- Not updating shared data structure.

Optional extensions (stay in Basics scope)

- Add update/delete option.

Quick check / exit ticket

Quick check: Why is a menu loop useful for CLI programs?

Hour 30: Input validation (Basics approach)

Outcomes

- Use simple guards to prevent crashes.
- Re-prompt on invalid input (basic).

Instructor talk points (10–15 min)

- Use .isdigit() for integer strings (limited but useful).
- Using if checks before conversion.
- Preview try/except (light) but save deep dive for Day 6.

Live demo (5–10 min)

34. Demo: safe int input function using while + isdigit().

Hands-on lab (25–35 min)

Lab: Safe Number Entry

- Create a loop that asks for a whole number.

- If input is invalid, show an error and re-prompt.

- Use the validated number in a small calculator.

Completion criteria

- Never crashes on non-numeric input.
- Accepts valid numbers correctly.

Common pitfalls to watch for

- isdigit() doesn’t handle negatives/decimals (mention limitation).

Optional extensions (stay in Basics scope)

- Extend to handle negative numbers (optional: strip leading '-').

Quick check / exit ticket

Quick check: What’s one limitation of isdigit()?

Hour 31: Mini-project: CLI Contact Manager (in-memory)

Outcomes

- Combine loops + dict/list into a functional CLI program.

- Practice clean UX messaging.

Instructor talk points (10–15 min)

- Define required features.
- Suggest incremental build: menu → add → list → search → delete.

Live demo (5–10 min)

35. Demo: implement add + list quickly; show testing strategy.

Hands-on lab (25–35 min)

Lab: Contact Manager (in-memory)

Required:

- Add contact (name, phone)

- List contacts

- Search by name

- Delete by name

Optional: update phone, normalize names

Use a looped menu until quit.

Completion criteria

- All required actions work.
- No KeyError for missing name.
- Clear prompts and confirmations.

Common pitfalls to watch for

- Case sensitivity and duplicates.
- Deleting while iterating.

Optional extensions (stay in Basics scope)

- Add: export to a formatted text block (still in-memory).

Quick check / exit ticket

Quick check: What was the hardest bug you hit in the menu loop?

Hour 32: Checkpoint 4: Control flow assessment

Outcomes

- Demonstrate conditionals + loops + data structures in a coherent program.

Instructor talk points (10–15 min)

- Explain rubric: correctness, readability, required constructs, robustness.

Live demo (5–10 min)

36. Demo: show how you'll spot-check quickly (run, try invalid input, try missing key).

Hands-on lab (25–35 min)

Checkpoint Lab 4 (45–60 min)

Build: CLI To-Do Manager

- Menu loop until quit.

- Add task, list tasks, mark complete, delete task.

- Store tasks in a list of dicts OR dict of status.

- Include basic input validation (non-empty task name).

Completion criteria

- Menu loop works.
- Tasks persist in memory during run.
- Complete/delete works.

Common pitfalls to watch for

- Losing data by reinitializing list inside loop.
- Index errors when deleting.

Optional extensions (stay in Basics scope)

- Add: filter by completed/uncompleted.

Quick check / exit ticket

Quick check: What's one strategy you used to test boundary cases?

Day 5

Session 9 (Hours 33–36)

Hour 33: Functions: def, parameters, return

Outcomes

- Write simple functions.
- Use return vs print appropriately.

Instructor talk points (10–15 min)

- Why functions: reuse + clarity.
- Parameters and arguments.
- Return values and storing results.

Live demo (5–10 min)

37. Demo: refactor a calculator into functions; show unit-like manual testing.

Hands-on lab (25–35 min)

Lab: Refactor Contact Manager

- Extract 3 functions: add_contact, list_contacts, search_contact.

- Each function should take data as input and return results or modify passed-in structure appropriately.

Completion criteria

- Functions called from menu.
- Less repeated code.

Common pitfalls to watch for

- Functions relying on globals unintentionally.
- Returning None by mistake.

Optional extensions (stay in Basics scope)

- Add: function to normalize name.
- Add: function to validate phone format (simple).

Quick check / exit ticket

Quick check: When should a function return a value instead of printing?

Hour 34: Scope + common function mistakes

Outcomes

- Explain local vs global scope.
- Use default parameters (simple cases).

Instructor talk points (10–15 min)

- Local variables exist only inside function.
- Avoid globals; pass data in.
- Default parameters for optional behavior.

Live demo (5–10 min)

38. Demo: show a bug caused by relying on global, then fix by passing list/dict.

Hands-on lab (25–35 min)

Lab: Calculator Functions

- Write functions: add, subtract, multiply, divide.

- Build a small CLI that calls them.

- Handle divide-by-zero with a simple if check (exceptions later).

Completion criteria

- Functions work and are called correctly.
- No crash on divide by zero.

Common pitfalls to watch for

- Shadowing variable names.
- Not returning the computed value.

Optional extensions (stay in Basics scope)

- Add: power and modulus.

Quick check / exit ticket

Quick check: If you create a variable inside a function, can you use it outside?

Hour 35: Functions with collections

Outcomes

- Pass lists/dicts into functions.
- Decide when to mutate vs return new collections.

Instructor talk points (10–15 min)

- Mutating a list vs returning a new list.
- Document expectations in docstrings.

Live demo (5–10 min)

39. Demo: normalize list of names; compare in-place vs new list.

Hands-on lab (25–35 min)

Lab: Normalize Contacts

- Write a function normalize_contacts(contacts) that strips and title-cases names.

- Demonstrate it on sample data.

- Print before/after clearly.

Completion criteria

- Names are normalized consistently.
- Learner can explain whether they mutated or returned.

Common pitfalls to watch for

- Accidentally reassigning local variable without returning.
- Modifying while iterating incorrectly.

Optional extensions (stay in Basics scope)

- Create a function to remove duplicates (basic).

Quick check / exit ticket

Quick check: What's one advantage of returning a new list instead of modifying in place?

Hour 36: Modules: imports and creating utils.py

Outcomes

- Import from standard library.
- Create your own module and import from it.

Instructor talk points (10–15 min)

- import vs from import.
- Common standard libs: math, random, datetime (light tour).
- Avoid circular imports (mention only).

Live demo (5–10 min)

40. Demo: create utils.py with 2 helper functions and import into main.py.

Hands-on lab (25–35 min)

Lab: Create a utils module

- Add: safe_int(prompt) and format_money(value).

- Use them inside an existing program (tip calc or to-do manager).

Completion criteria

- Module imports correctly.
- Helpers used at least twice.

Common pitfalls to watch for

- Running wrong file as main module.
- Name conflict with standard library (e.g., naming file 'random.py').

Optional extensions (stay in Basics scope)

- Add a third helper: menu_choice(prompt, options).

Quick check / exit ticket

Quick check: What’s the difference between import math and from math import sqrt?

Session 10 (Hours 37–40)

Hour 37: Intro classes: objects, attributes, methods

Outcomes

- Create a basic class with __init__.
- Instantiate objects and call methods.

Instructor talk points (10–15 min)

- Class vs object.
- Attributes and methods.
- Keep it simple: one class, no inheritance.

Live demo (5–10 min)

41. Demo: Contact class with name/phone; display() method.

Hands-on lab (25–35 min)

Lab: Contact class

- Create a Contact class with name and phone.

- Add a display() method that returns a formatted string.

- Store multiple Contact objects in a list and print them.

Completion criteria

- Objects created and stored.
- display() works.

Common pitfalls to watch for

- Forgetting self parameter.
- Mixing up class attribute vs instance attribute.

Optional extensions (stay in Basics scope)

- Add: update_phone(new_phone) method with simple validation.

Quick check / exit ticket

Quick check: What does self represent?

Hour 38: Collections of objects + searching

Outcomes

- Search lists of objects.
- Update object fields safely.

Instructor talk points (10–15 min)

- Loop through objects and compare attributes.
- Finding by name (normalize).
- Updating fields.

Live demo (5–10 min)

42. Demo: search_contact() returning an object; then update it.

Hands-on lab (25–35 min)

Lab: Refactor Contact Manager to objects

- Replace dict-based contacts with Contact objects.

- Implement search by name.

- Implement update phone.

- Keep menu-driven flow.

Completion criteria

- Program still functions with object storage.
- Update works.

Common pitfalls to watch for

- Comparing wrong attribute.
- Multiple contacts with same name (decide rule).

Optional extensions (stay in Basics scope)

- Add: allow duplicate names by adding an ID field (simple integer counter).

Quick check / exit ticket

Quick check: Why might objects make code easier to maintain than nested dicts?

Hour 39: Basic encapsulation + data validation

Outcomes

- Use simple validation inside methods.
- Keep business rules in one place.

Instructor talk points (10–15 min)

- Validation as guardrails.
- Raising simple ValueError (preview; full exception handling later).
- Keeping UI separate from core logic.

Live demo (5–10 min)

43. Demo: Contact.update_phone checks length/digits; if invalid, returns False or raises ValueError.

Hands-on lab (25–35 min)

Lab: Add validation

- Add a validation rule to at least one class method.

- Update your menu to handle invalid entries gracefully (message + re-prompt).

Completion criteria

- Invalid input handled without crashing.
- Validation implemented in class or helper.

Common pitfalls to watch for

- Overly strict rules block valid formats.
- Forgetting to handle return/exception in UI.

Optional extensions (stay in Basics scope)

- Add: normalize phone to digits-only for storage.

Quick check / exit ticket

Quick check: Where is the best place to enforce validation: in UI code or in the class?

Hour 40: Checkpoint 5: Functions + modules + intro OOP

Outcomes

- Demonstrate modular code organization and at least one class.

Instructor talk points (10–15 min)

- Explain deliverables: main.py + utils.py + class file (optional).
- Remind: readable prompts and output.

Live demo (5–10 min)

44. Demo: run through a checklist of required features.

Hands-on lab (25–35 min)

Checkpoint Lab 5 (45–60 min)

Build: CLI Organizer (in-memory)

- Menu loop until quit.

- At least 3 functions.

- At least 1 custom module.

- At least 1 class used to store a record (Task or Contact).

- Features: add, list, search.

No file I/O yet.

Completion criteria

- Meets all requirements.
- Runs without crashes on typical inputs.
- Code is organized across files.

Common pitfalls to watch for

- Import errors due to wrong working directory.
- Creating module with same name as stdlib.

Optional extensions (stay in Basics scope)

- Add: delete/update feature if time.

Quick check / exit ticket

Quick check: Ask learners to point to one place they reduced code duplication with functions.

Day 6

Session 11 (Hours 41–44)

Hour 41: Files: reading and writing text

Outcomes

- Open files using with.
- Read lines and write lines.

Instructor talk points (10–15 min)

- File paths and working directory.
- with open(...) as f.
- read(), readlines(), write().

Live demo (5–10 min)

45. Demo: write a shopping list to file; then read and print.

Hands-on lab (25–35 min)

Lab: Save and Load (text)

- Write your contacts/tasks to a text file (one per line).

- Load them back and print.

- Keep format simple (e.g., name|phone).

Completion criteria

- File created and reloaded correctly.
- Learner can locate file on disk.

Common pitfalls to watch for

- Wrong path / wrong working directory.
- Forgetting newline characters.

Optional extensions (stay in Basics scope)

- Add: timestamp header line.
- Add: export pretty report file.

Quick check / exit ticket

Quick check: Why is using with recommended for file handling?

Hour 42: JSON persistence (stdlib json)

Outcomes

- Serialize data to JSON.
- Load JSON back into Python structures.

Instructor talk points (10–15 min)

- json.dump/json.load basics.
- What JSON can store (dict/list/str/num/bool/null).
- Mapping objects to dicts (simple to_dict/from_dict).

Live demo (5–10 min)

46. Demo: dump dict/list to data.json; load; print.
47. If using objects: show to_dict().

Hands-on lab (25–35 min)

Lab: Save and Load (JSON)

- Export your app's data to data.json.

- On startup, try to load it (if present).

- After changes, save again.

Completion criteria

- Data persists across runs.
- No crash when file missing (use if-check or try/except).

Common pitfalls to watch for

- JSONDecodeError from corrupted file.
- Trying to dump custom objects directly.

Optional extensions (stay in Basics scope)

- Add: backup previous JSON before overwrite.
- Add: pretty-print JSON with indent=2.

Quick check / exit ticket

Quick check: Why can't json.dump() write a custom object unless you convert it first?

Hour 43: Directories + paths (pathlib preferred)

Outcomes

- Use pathlib to build reliable paths.
- Create a data directory if needed.

Instructor talk points (10–15 min)

- Relative vs absolute.
- Path.cwd() and Path('data')
- mkdir(exist_ok=True).

Live demo (5–10 min)

48. Demo: create data/ folder and write file into it using pathlib.

Hands-on lab (25–35 min)

Lab: Data folder

- Create a data/ folder.

- Save JSON into data/data.json.

- Make sure your program runs regardless of where the terminal starts (within reason).

Completion criteria

- Uses pathlib or os.path to target data/.
- Folder is created safely.

Common pitfalls to watch for

- Hard-coded absolute paths.
- Running from a different working directory breaks imports/paths.

Optional extensions (stay in Basics scope)

- Add: allow user to choose file name.

Quick check / exit ticket

Quick check: What's the difference between a relative and absolute path?

Hour 44: Exception handling (full Basics treatment)

Outcomes

- Use try/except to catch common runtime errors.
- Catch specific exceptions when practical.

Instructor talk points (10–15 min)

- try/except/else/finally patterns.
- Catching ValueError for conversions.
- FileNotFoundError and JSONDecodeError.

Live demo (5–10 min)

49. Demo: wrap numeric input and file loading; show friendly error messages.

Hands-on lab (25–35 min)

Lab: Harden your program

- Add try/except around:

1) numeric input conversions

2) JSON load

- On error, show a friendly message and continue running.

- Keep exception handling specific when possible (ValueError, FileNotFoundError).

Completion criteria

- Program continues after bad input.
- Handles missing/corrupt file gracefully.

Common pitfalls to watch for

- Catching Exception broadly everywhere (discourage).
- Swallowing errors without message.

Optional extensions (stay in Basics scope)

- Log errors to a text file (simple).

Quick check / exit ticket

Quick check: Why is catching a specific exception better than catching all exceptions?

Session 12 (Hours 45–48)

Hour 45: Capstone kickoff: requirements + scaffolding

Outcomes

- Define capstone requirements and plan steps.
- Build project skeleton and first working flow.

Instructor talk points (10–15 min)

- Capstone spec review.
- Minimum viable product first.
- File layout: main.py + modules.

Live demo (5–10 min)

50. Demo: scaffold menu + load data + one action + save.

Hands-on lab (25–35 min)

Capstone: CLI Personal Organizer

Choose a theme:

- Tasks

- Contacts

- Notes

Required:

- Menu loop

- Functions split into modules

- At least one class

- JSON persistence in data/ folder

- Exception handling for input and file errors

Deliverable this hour: skeleton + load/save + one CRUD action working.

Completion criteria

- Repo/folder created.
- Program runs and shows menu.
- Load/save works even if file missing.

Common pitfalls to watch for

- Overbuilding before MVP works.
- Complex data model too early.

Optional extensions (stay in Basics scope)

- Add a search feature early (basic) to keep capstone useful.

Quick check / exit ticket

Quick check: What's your MVP feature set?

Hour 46: Capstone build sprint

Outcomes

- Implement full CRUD feature set.
- Improve UX and robustness.

Instructor talk points (10–15 min)

- Iterative building: add → list → search → update/delete.
- Testing as you go.

Live demo (5–10 min)

51. Demo: implement one feature end-to-end with save/load.

Hands-on lab (25–35 min)

Capstone build

- Complete required features.

- Add at least one 'quality' improvement:

- sorted output

- confirmations

- input validation

- Instructor circulates and spot-checks.

Completion criteria

- All required features implemented.

- Data persists across runs.

Common pitfalls to watch for

- Saving not called after updates.
- Data structure mismatch between save and load.

Optional extensions (stay in Basics scope)

- Add simple report export (text file).

Quick check / exit ticket

Quick check: What's one bug you fixed today and how?

Hour 47: Capstone polish + demo readiness

Outcomes

- Finalize capstone quality.
- Prepare a short demo walkthrough.

Instructor talk points (10–15 min)

- Polish checklist: prompts, help text, edge cases.
- Readme notes (how to run).

Live demo (5–10 min)

52. Demo: show a 'good demo' flow and how to handle a mistake live.

Hands-on lab (25–35 min)

Capstone polish

- Add README.txt or README.md with run instructions.

- Add sample data file (optional).

- Run through a full demo script: start → add → list → search → save → restart → load.

Completion criteria

- Runs cleanly from scratch.
- Demo flow works in <3 minutes.

Common pitfalls to watch for

- Last-minute refactors break working code.
- Forgetting to test from a fresh run.

Optional extensions (stay in Basics scope)

- Add a help screen option in menu.

Quick check / exit ticket

Quick check: What will you show first in your demo and why?

Hour 48: Final assessment + certification-style review

Outcomes

- Demonstrate end-to-end competency in Basics objectives.
- Identify next-step study targets.

Instructor talk points (10–15 min)

- Review key topics: types, structures, flow, functions, modules, classes, files, exceptions.
- Exam-style question patterns (conceptual + code reading).

Live demo (5–10 min)

53. Demo: quick ‘read this code and predict output’ exercise.

Hands-on lab (25–35 min)

Final (60–75 min)

1) Capstone demo + rubric scoring

2) Short written/quiz review (15–20 min)

3) Individual feedback: strengths + what to practice next

Completion criteria

- Capstone meets requirements.
- Learner can explain their code at a high level.

Common pitfalls to watch for

- Time management; leaving demo prep too late.

Optional extensions (stay in Basics scope)

- Optional: provide a practice exam link or printed question set (no new topics).

Quick check / exit ticket

Quick check: Ask each learner for one concept they feel strong about and one to review.

Checkpoint & capstone grading rubrics

Checkpoint rubric (use for Checkpoints 1–5)

Score each category 0–3. Suggested pass threshold: 8/12 (adjust to your program requirements).

Capstone rubric (final)

Troubleshooting quick guide (Basics)

Pacing adjustments

If learners are ahead: use the ‘Optional extensions’ for each hour and have them explain tradeoffs.

If learners are behind: shorten lecture, do one extra guided example, and reduce lab scope to MVP.

Encourage pair debugging during checkpoints only if your program allows it; otherwise pair practice in non-assessment hours.

No-go topics for the Basics course (keep for Advanced)

Web frameworks (Flask/Django), REST APIs

Databases/SQL/ORM

GUI frameworks (Tkinter/PyQt)

Testing frameworks (pytest), coverage tooling

Packaging/distribution (wheels, publishing), deployment

Data science stack (numpy/pandas/matplotlib/ML)

Checkpoint & capstone grading rubrics

Checkpoint rubric (use for Checkpoints 1–5)

Score each category 0–3. Suggested pass threshold: 8/12 (adjust to your program requirements).

Category	What you're looking for
Correctness	Meets the functional requirements; produces correct output for typical test cases.
Required concepts	Uses the concepts targeted for that checkpoint (e.g., lists, loops) in an appropriate way.
Readability	Clear variable names, consistent formatting, helpful comments where needed.
Robustness (Basics)	Handles at least a few invalid/missing inputs gracefully; avoids obvious crashes.

Capstone rubric (final)

Category	What you're looking for
Requirements met	Menu loop, functions split into modules, at least one class, JSON persistence, exception handling.
User experience	Clear prompts, confirmations, and readable output.
Code organization	Logical file structure; functions/classes have clear responsibility.
Data persistence	Save/load works across runs; handles missing/corrupt file gracefully.
Demo & explanation	Learner can explain their approach and show key features in <3 minutes.

Troubleshooting quick guide (Basics)

Symptom	Fast fix
IndentationError	Check inconsistent tabs/spaces; reindent the block with spaces (4).
NameError	Variable misspelling or used before assignment; check exact casing.
TypeError (can't add str + int)	Convert input to int/float before math; use f-strings for output.
ValueError on int()/float()	Input wasn't numeric; add validation or try/except and re-prompt.
IndexError	Index out of range; print len(list) and check boundaries.
KeyError	Missing dict key; use 'in' check or dict.get().
FileNotFoundError	Wrong path; print current working directory; use pathlib and create data/ folder.
JSONDecodeError	Corrupted JSON; catch it and reset to defaults or restore from backup.

Pacing adjustments

- If learners are ahead: use the 'Optional extensions' and have them explain tradeoffs.
- If learners are behind: shorten lecture, do one extra guided example, and reduce lab scope to MVP.
- For 4-hour sessions: consider assigning optional extensions as homework instead of in-session work.

No-go topics for the Basics course (keep for Advanced)

- Web frameworks (Flask/Django), REST APIs
- Databases/SQL/ORM
- GUI frameworks (Tkinter/PyQt)
- Testing frameworks (pytest), coverage tooling
- Packaging/distribution (wheels, publishing), deployment
- Data science stack (numpy/pandas/matplotlib/ML)