Python Programming — Basics (48 Hours)

Student-facing catalog syllabus (one page)

Course description

This 48-hour, instructor-led course builds practical Python programming ability from the ground up. Learners write and debug multiple programs while learning core syntax and control flow, collections, and functions/modules. The course also introduces exceptions and file-based persistence, plus an intro to object-oriented programming (OOP). It culminates in a capstone project and final assessment.

Audience

Learners new to Python (or early-stage) who want hands-on programming fundamentals for workplace scripting, continued study, and entry-level certification readiness.

Prerequisites

Basic computer proficiency. Prior programming exposure is helpful but not required.

Duration and schedule

48 hours total (typical delivery: **6 days × 8 hours/day**). Alternative formats (same total hours): **12 sessions × 4 hours** or **8 sessions × 6 hours**.

Learning outcomes

By the end of this course, learners will be able to: - **Build** and debug Python programs, including reading and interpreting tracebacks. - **Control** program execution using variables, expressions, console I/O, decisions, and loops. - **Work with** built-in data structures (lists, tuples, sets, dictionaries) to store and process data. - **Organize** solutions using functions, modules, and basic package imports. - **Persist and handle errors** by reading/writing files and using exceptions with basic input validation. - **Model with OOP** by creating simple classes and objects.

Module outline

1. **Getting Started with Python** — write first scripts
2. **Controlling Execution and Basic I/O** — if/loops + input
3. **Using Data Structures** — lists, tuples, sets, dicts
4. **Processing Data Structures** — iterate and transform data
5. **Creating Functions** — reuse logic with functions
6. **Using Packages** — imports and code organization
7. **Error-Handling and Data Persistence** — exceptions + files
8. **Object-Oriented Programming (Basics)** — classes and objects

Assessment

- Guided practice and labs throughout
- Periodic knowledge/skill checkpoints
- **Capstone project** (menu-driven Personal Organizer application)
- Final assessment and review

Certification alignment (Python Institute)

- **PCEP**: Strong alignment across fundamentals, control flow, collections, functions, and exceptions.
- **PCAP**: Introductory alignment (modules/packages, exceptions, strings, OOP, and file I/O); deeper PCAP breadth is typically reinforced in the follow-on Advanced course.

Instructor-facing internal syllabus (checkpoint names + capstone rubric)

Course structure (runbook-aligned)

- **Day 1**: Orientation + fundamentals; intro decisions/loops → **Checkpoint 1**
- **Day 2**: Deeper looping + debugging; begin lists and iteration patterns → **Checkpoint 2**
- **Day 3**: Data structures in depth + operations across structures → **Checkpoint 3**
- **Day 4**: Functions + modularity; intro persistence and exception handling foundations → **Checkpoint 4 (Functions/Modularity)**
- **Day 5**: OOP basics + program organization; integrate prior topics in a multi-part program → **Checkpoint 5 (OOP/Integration)**
- **Day 6**: Certification-style review + capstone build + **Final assessment**

Checkpoints (formative)

Use checkpoints to verify readiness before moving forward. Expected checkpoint coverage: - **Checkpoint 1**: Syntax/variables/types/operators/I-O; basic decisions/loops - **Checkpoint 2**: Looping fluency, debugging patterns, early list usage - **Checkpoint 3**: Lists/tuples/sets/dicts operations; iteration across structures - **Checkpoint 4**: Functions (parameters/returns), decomposition, module organization - **Checkpoint 5**: OOP basics + integrating prior topics into a multi-part program (as scheduled)

Capstone project: “Personal Organizer” (summative)

Goal: Menu-driven organizer app demonstrating integrated skills.

Required features (minimum acceptance criteria)

- **Menu-driven UI flow** with clear options and loop-based navigation
- **Input validation** (handles empty/invalid entries gracefully)
- **Data management using collections** (e.g., list/dict to store items)
- **Functional decomposition** (functions for add/edit/delete/search/list/save/load)
- **Persistence** (save/load organizer data using a file format such as JSON or CSV)

- **At least one class** to model organizer entities and/or behaviors
- **Error handling** using exceptions for common failure paths (bad input, missing file, parse errors)

Suggested rubric (100 points)

1. **Program correctness & completeness (30 pts)**
2. All required features present and functional; meets stated behaviors.
3. **Usability & validation (15 pts)**
4. Clear prompts, resilient input handling, user-friendly messages.
5. **Data structures & logic (15 pts)**
6. Appropriate use of collections; efficient and readable operations.
7. **Functions & modularity (15 pts)**
8. Logical decomposition; minimal duplication; clean organization.
9. **Persistence implementation (10 pts)**
10. Save/load works reliably; handles missing/corrupt data gracefully.
11. **OOP application (10 pts)**
12. Class design appropriate for the problem; sensible attributes/methods.
13. **Code quality (5 pts)**
14. Readability, naming, formatting, inline comments where needed.

Final assessment (summative)

- Short certification-style review block + practical verification using capstone demonstration and/or a brief skills check.

Notes for delivery

- Keep pacing lab-forward: short concept blocks followed by guided practice.
- Encourage incremental builds (start small, test often) to reduce cognitive load.
- Reserve Day 6 time explicitly for capstone completion and review.