Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: PYTH • **Updated:** 19 Nov 2026
https://mockattempt.com/courses/python-programming-training

PYTH • PROGRAMMING & BACKEND | **BEGINNER TO ADVANCED** | **100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER**

Core & Advanced Python 3.12 Programming

Go from zero coding background to writing clean, idiomatic, tested Python 3.12 software mastering data structures, OOP, decorators, generators, context managers, AsyncIO concurrency, virtual environments, and pytest.

Duration: 7 Weeks • Weekend Live (Sat–Sun • 2h/Day) (80 Hrs) • **Modules:** 7 • **Live Topics:** 116 • **Subtopics:** 928+ • **Labs & Tasks:** 232 • **Mocks / AI Vivas:** 116 / 116 • **Projects:** 7 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹1,999 (Net Tuition ₹1,694 + Included 18% GST ₹305) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹25/Guided Hr (~₹25/Live Topic)

Faculty: 100% Active Working IT Professionals • **Weekend-Only Live Batch (7:00 AM – 8:00 PM IST Flexible Slots):** Weekend-Only Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday Only (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Weekend Slot Based on Working Professional Trainer Availability) • 10–12 Weeks Weekend Track)

✓ **Trainer LeetCode & HackerRank Mentorship (Python 3.12): Live trainer support to crack LeetCode & HackerRank problems, optimize complexity, and handle edge cases.**

Core Technologies & Engineering Stack: Python 3.12 • Lists, Dicts & Sets • Functions & Type Hints • OOP & @dataclass • Decorators & Generators • Exceptions & Context Mgrs • File, CSV & JSON I/O • Threading & AsyncIO • venv, uv & Packaging • Pytest & Git

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Write clean, type-hinted Python 3.12 applications using `list`/`dict`/`set` comprehensions, closures, and decorators.
2. Model domain systems using 4-pillar Python OOP, `@dataclass`, abstract base classes (`ABC`), and dunder protocols.
3. Process large files lazily with generators (`yield`), context managers, structured logging, and `asyncio` coroutines.
4. Package reproducible Python projects with `venv`/`uv`, automated `pytest` suites, and clean GitHub workflows.
5. Crack LeetCode and HackerRank problems in Python 3.12 with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
6. Automated Testing, Debugging & Quality Gates (Python 3.12, Lists, Dicts & Sets, Functions & Type Hints, OOP & @dataclass, Decorators & Generators) — Automated Python 3.12, Lists, Dicts & Sets, Functions & Type Hints, OOP & @dataclass, Decorators & Generators test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
7. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Core & Advanced Python 3.12 Programming topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
8. Translated live client functional and non-functional requirements into modular Python 3.12, Lists, Dicts & Sets, Functions & Type Hints, OOP & @dataclass, Decorators & Generators production architecture.
9. Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
10. Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
11. Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.
12. Clear 100% topic-wise Timed MCQ Mock Tests and Spoken MockAttempt AI Technical Interviews across every module of Core & Advanced Python 3.12 Programming.

Eligibility & Prerequisites: Zero prior programming experience required — Week 1 begins from installing Python 3.12 and writing your first script. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend Weekend-Only live classes (Saturday & Sunday, 2 Hours/Day between 7:00 AM – 8:00 PM IST taught by Working IT Professionals), complete weekday real-repository checklists, and practice topic-wise Mock Tests & AI Interviews. • Dedication to write original code in your GitHub portfolio and complete the mentor-guided Live Client Production Sprint.

2. DETAILED ZERO-TO-EXPERT MODULE, TOPIC & SUBTOPIC SYLLABUS (7 MODULES • 116 LIVE TOPICS • 928+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Month 1 – Core Python 3.12, Data Structures, Functions & OOP (Weeks 1–4) (70 Topics / 560 Subtopics) • Modules 5–7: Month 2 – Decorators, Generators, AsyncIO, Pytest & Capstone (Weeks 5–7) (46 Topics / 368 Subtopics)

MODULE 1 OF 7 • STAGE 1 • 11 LIVE + LAB HOURS • 16 TOPICS (128 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Python 3.12 Setup, Execution Model, Variables & Control Flow

Master algorithmic thinking, Python 3.12 installation on Windows/Linux, VS Code & PyCharm setup, interactive vs script execution, indentation syntax, variables, dynamic typing, type conversion, console I/O, arithmetic/bitwise/membership/identity operators, conditionals, `for`/`while` loops, `range()`, pattern programming, and debugging.

Objectives: ✓ Install Python 3.12 cleanly on Windows and Linux, configure VS Code/PyCharm, and execute code in REPL and script modes. • ✓ Declare variables and apply `int`, `float`, `complex`, `bool`, `str`, and explicit type conversions safely. • ✓ Evaluate expressions accurately using arithmetic, comparison, logical, bitwise, membership (`in`), identity (`is`), and precedence rules. • ✓ Control program flow with `if`/`elif`/`else`, nested branches, `for`/`while` loops, `range()`, and `break`/`continue`/`pass`. • ✓ Diagnose `SyntaxError`, `IndentationError`, `TypeError`, and `ValueError` while building 7 practical CLI programs.

1.1 Introduction to Programming [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Programming concepts • Step-by-step algorithms • Flowcharts • Dry-running • Production Pitfall Guard (Introduction to Programming) • Hands-On Verification (Introduction to Programming) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Introduction to Programming

Purpose: Breaking complex backend business rules (like tiered utility billing) into deterministic input -> process -> output steps before coding.

What Topic Is Next → Next Topic 1.2: Python Introduction

1.2 Python Introduction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Python history • Readability philosophy (PEP 8 / Zen of Python) • Backend/API/automation/data applications • The Python 3 ecosystem • Production Pitfall Guard (Python Introduction) • Hands-On Verification (Python Introduction) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Introduction

Purpose: Choosing Python 3.12 + Django/FastAPI for rapid, high-maintainability REST APIs, microservices, and background task pipelines.

What Topic Is Next → Next Topic 1.3: Python Installation

1.3 Python Installation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing Python 3.12 on Windows (`PATH` & `py` launcher) • Linux (`apt`/`pyenv`) • Verifying `python3` • `pip` • Production Pitfall Guard (Python Installation) • Hands-On Verification (Python Installation) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Installation

Purpose: Maintaining consistent Python 3.12 interpreter versions across local Windows laptops and Ubuntu Linux deployment servers.

What Topic Is Next → Next Topic 1.4: Environment Setup

1.4 Environment Setup [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring VS Code (Python + Pylance + Ruff extensions) • PyCharm • Integrated terminal • Selecting the workspace interpreter • Production Pitfall Guard (Environment Setup) • Hands-On Verification (Environment Setup) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Environment Setup

Purpose: Catching undefined variables, bad imports, and PEP 8 formatting issues instantaneously inside VS Code or PyCharm.

What Topic Is Next → Next Topic 1.5: Python Execution

1.5 Python Execution [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Interactive REPL mode vs `.py` script mode • Source -> bytecode (`.pyc` in `__pycache__`) -> Python Virtual Machine (PVM) execution flow • Production Pitfall Guard (Python Execution) • Hands-On Verification (Python Execution) • Python Execution — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Execution • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Testing quick expressions in the REPL while structuring repeatable backend tools as executable `.py` modules.

What Topic Is Next → Next Topic 1.6: Python Syntax

1.6 Python Syntax [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 4-space block indentation rules (PEP 8) • Logical vs physical lines • Docstrings • Meaningful single-line `#` comments • Production Pitfall Guard (Python Syntax) • Hands-On Verification (Python Syntax) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Syntax

Purpose: Enforcing consistent 4-space PEP 8 formatting across multi-developer Django and FastAPI codebases.

What Topic Is Next → Next Topic 1.7: Variables

1.7 Variables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dynamic variable binding • Object references • `snake_case` naming rules • Multiple assignment • Production Pitfall Guard (Variables) • Hands-On Verification (Variables) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Variables

Purpose: Binding descriptive domain names (`net_payable_inr`, `is_verified`) and unpacking coordinate/pair values cleanly.

What Topic Is Next → Next Topic 1.8: Data Types

1.8 Data Types [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Built-in scalar types (`int` arbitrary precision • `float` IEEE-754 • `complex` • `bool` • Production Pitfall Guard (Data Types) • Hands-On Verification (Data Types) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Data Types

Purpose: Representing integer paise/cents accurately with `int` or `Decimal` instead of binary `float` rounding drift (`0.1 + 0.2`).

What Topic Is Next → Next Topic 1.9: Type Conversion

1.9 Type Conversion [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implicit numeric promotion (`int` -> `float`) vs explicit constructors (`int()` • `float()` • `str()` • `bool()`) • Production Pitfall Guard (Type Conversion) • Hands-On Verification (Type Conversion) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Type Conversion

Purpose: Parsing string query parameters and CLI `input()` strings into validated numeric integers and floats.

What Topic Is Next → Next Topic 1.10: Input and Output

1.10 Input and Output [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading console text via `input()` • `print()` (`sep` • `end` • `flush`) • Production Pitfall Guard (Input and Output) • Hands-On Verification (Input and Output) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Input

Purpose: Formatting aligned CLI invoice tables and diagnostic logs with precision specifiers (`f"{price:,.2f}"`).

What Topic Is Next → Next Topic 1.11: Operators

1.11 Operators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Arithmetic (`+` • `-` • `*` • `/` true division • Production Pitfall Guard (Operators) • Hands-On Verification (Operators) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Operators

Purpose: Computing pagination total pages `(total + page_size - 1) // page_size` and short-circuiting guard conditions.

What Topic Is Next → Next Topic 1.12: Bitwise Operators

1.12 Bitwise Operators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bitwise • (`&`) • OR (`|`) • XOR (`^`) • Production Pitfall Guard (Bitwise Operators) • Hands-On Verification (Bitwise Operators) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Bitwise Operators

Purpose: Packing compact permission bitmasks (`READ = 1`, `WRITE = 2`, `ADMIN = 4`) and fast parity/power-of-two checks.

What Topic Is Next → Next Topic 1.13: Membership Operators

1.13 Membership Operators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `in` • `not in` across strings • Lists • Tuples • Production Pitfall Guard (Membership Operators) • Hands-On Verification (Membership Operators) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Membership Operators

Purpose: Validating allowed HTTP roles or file extensions in `O(1)` average time using a constant `set` (`ext in ALLOWED_EXTS`).

What Topic Is Next → Next Topic 1.14: Identity Operators

1.14 Identity Operators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `is` • `is not` • `id()` memory address checks • Singleton `None` checks (`x is None`) • Production Pitfall Guard (Identity Operators) • Hands-On Verification (Identity Operators) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Identity Operators

Purpose: Writing PEP 8 compliant `if user_id is None:` checks that cannot be fooled by custom `__eq__` overrides or `0`/`""` falsy values.

What Topic Is Next → Next Topic 1.15: Operator Precedence

1.15 Operator Precedence [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Python evaluation order (`**` • Unary `-` • `* / // %` • `+ -` • Production Pitfall Guard (Operator Precedence) • Hands-On Verification (Operator Precedence) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Operator Precedence

Purpose: Writing readable range validations like `if 0 <= marks <= 100:` and explicit parentheses in financial formulas.

What Topic Is Next → Next Topic 1.16: Conditional Statements

1.16 Conditional Statements [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `if` • `elif` • `else` branching • Mutually exclusive grade/slab ladders • Production Pitfall Guard (Conditional Statements) • Hands-On Verification (Conditional Statements) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Conditional Statements

Purpose: Implementing tiered Electricity Bill slabs and Student Grade classification (`A+`, `A`, `B`, `C`, `Fail`).

What Topic Is Next → Module 1 Capstone (Week 1 Lab Build: 7-in-1 Python CLI Utility Suite) → Next Topic 2.1: Strings

Module 1 Hands-On Capstone Project (100 Marks): Week 1 Lab Build: 7-in-1 Python CLI Utility Suite — Build a PEP 8 compliant Python 3.12 command-line utility suite combining Calculator, Simple Interest Calculator, Tiered Electricity Bill Calculator, Student Grade Calculator, O(sqrt(n)) Prim...

MODULE 2 OF 7 • STAGE 1 • 11 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Strings, Lists, Tuples, Sets, Dict Hashing & Comprehensions

Master every built-in Python data structure and text-processing API: string indexing, slicing, methods, searching, validation, f-strings, immutability, lists, list comprehensions, tuples, sets, set algebra, dictionaries, dict comprehensions, nested collections, mutability, shallow vs deep copy, and Big-O structure selection.

Objectives: ✓ Manipulate and validate text cleanly using string slicing, searching, transformation methods, and f-strings. • ✓ Build and transform dynamic sequences with `list` methods, sorting keys, and readable list comprehensions. • ✓ Leverage immutable `tuple` records, `O(1)` `set` algebra (`|`, `&`, `-`), and

`dict` hash tables (`get`, `setdefault`, comprehensions). • ✓ Model real backend payloads using nested lists/dicts while preventing aliasing bugs via `copy()` vs `copy.deepcopy()`. • ✓ Select the optimal collection (`list` vs `tuple` vs `set` vs `dict`) based on mutability and Big-O lookup cost.

2.1 Strings [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unicode UTF-8 string literals • Raw strings `r"..."` • Multiline strings • Positive `0..n-1` • Production Pitfall Guard (Strings) • Hands-On Verification (Strings) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Strings

Purpose: Parsing UTF-8 names, emails, slugs, and regex raw patterns reliably in internationalized backends.

What Topic Is Next → Next Topic 2.2: String Slicing

2.2 String Slicing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Slice notation `s[start:stop:step]` • Omitting bounds • Negative steps `s[::-1]` reversal • Out-of-range slice safety • Production Pitfall Guard (String Slicing) • Hands-On Verification (String Slicing) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for String Slicing

Purpose: Masking sensitive PAN/card digits (`"*" * 8 + card[-4:]`) and extracting fixed-width transaction prefixes.

What Topic Is Next → Next Topic 2.3: String Methods

2.3 String Methods [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `upper()` • `lower()` • `casefold()` • `strip()`/`lstrip()`/`rstrip()` • Production Pitfall Guard (String Methods) • Hands-On Verification (String Methods) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for String Methods

Purpose: Normalizing user emails (`email.strip().lower()`), extracting `Bearer ` JWT tokens via `removeprefix("Bearer ")`, and building CSV lines with `",".join()`.

What Topic Is Next → Next Topic 2.4: String Searching

2.4 String Searching [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `find()` / `rfind()` (`-1` on miss) vs `index()` / `rindex()` (`ValueError` on miss) • Non-overlapping `count()` • Production Pitfall Guard (String Searching) • Hands-On Verification (String Searching) • String Searching — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for String Searching • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Locating delimiter positions in log lines (`line.partition(":")` or `find()`) without raising unhandled exceptions.

What Topic Is Next → Next Topic 2.5: String Validation

2.5 String Validation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `startswith()` (accepting a tuple of prefixes!) • `endswith()` • `isdigit()` • `isalpha()` • Production Pitfall Guard (String Validation) • Hands-On Verification (String Validation) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for String Validation

Purpose: Validating HTTPS URLs `url.startswith(("https://", "http://"))` and allowed upload extensions `filename.lower().endswith((".pdf", ".docx"))`.

What Topic Is Next → Next Topic 2.6: String Formatting

2.6 String Formatting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern f-strings `f"{expr}"` • Debug `f"{var=}"` • Numeric formatting `:.2f` • ISO dates • Production Pitfall Guard (String Formatting) • Hands-On Verification (String Formatting) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for String Formatting

Purpose: Rendering clean notification templates, currency outputs, and debug assertions (`f"{order_id=} {status=}"`).

What Topic Is Next → Next Topic 2.7: String Immutability

2.7 String Immutability [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `str` objects cannot be mutated in-place • `O(n^2)` loop `+=` concatenation penalty vs `O(n)` `"".join(parts)` buffer building • Production Pitfall Guard (String Immutability) • Hands-On Verification (String Immutability) • String Immutability — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for String Immutability • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Assembling 50,000-row CSV exports in `O(n)` time via `"\n".join(rows)` instead of quadratic string concatenation.

What Topic Is Next → Next Topic 2.8: Lists

2.8 Lists [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dynamic array internals of `list` • Literal/constructor creation • Indexing • Slicing • Production Pitfall Guard (Lists) • Hands-On Verification (Lists) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Lists

Purpose: Storing ordered query results, paginated items, and mutable batch buffers.

What Topic Is Next → Next Topic 2.9: List Operations

2.9 List Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `append()` (`O(1)` amortized) • `extend()` • `insert(0 • X)` (`O(n)` shift!) • Production Pitfall Guard (List Operations) • Hands-On Verification (List Operations) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for List Operations

Purpose: Accumulating validated records with `append()`/`extend()` while avoiding `O(n)` head insertions.

What Topic Is Next → Next Topic 2.10: List Methods

2.10 List Methods [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `pop()` (`O(1)` tail vs `O(n)` index) • `clear()` • In-place `sort(key=... • Reverse=...)` vs non-mutating `sorted()` • Production Pitfall Guard (List Methods) • Hands-On Verification (List Methods) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for List Methods

Purpose: Sorting employee records by `(-salary, name)` using Timsort (`O(n log n)` stable sort) without mutating caller inputs.

What Topic Is Next → Next Topic 2.11: List Comprehension

2.11 List Comprehension [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `[expr for x in iterable if cond]` • Conditional transformations • Flattening 2D matrices • Readability limits • Production Pitfall Guard (List Comprehension) • Hands-On Verification (List Comprehension) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for List Comprehension

Purpose: Transforming ORM objects into clean response dictionaries or filtering active SKUs in one concise expression.

What Topic Is Next → Next Topic 2.12: Tuples

2.12 Tuples [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable fixed-structure `tuple` records • Single-element trailing comma `(42 • Tuple packing/unpacking • Hashability as `dict` keys • Production Pitfall Guard (Tuples) • Hands-On Verification (Tuples) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Tuples

Purpose: Returning `(status_code, payload)` pairs from functions and using `(tenant_id, user_id)` composite keys in cache dictionaries.

What Topic Is Next → Next Topic 2.13: Tuple Methods

2.13 Tuple Methods [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.count(value)` • `.index(value)` • Memory efficiency of tuples vs lists (`sys.getsizeof`) • Immutability of slot bindings • Production Pitfall Guard (Tuple Methods) • Hands-On Verification (Tuple Methods) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Tuple Methods

Purpose: Storing constant configuration choices (`STATUS_CHOICES`) that accidental caller code cannot `.append()` to.

What Topic Is Next → Next Topic 2.14: Sets

2.14 Sets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Hash-table backed `set` • Immutable `frozenset` • `set()` empty set creation (NOT `{}`!) • `O(1)` membership • Production Pitfall Guard (Sets) • Hands-On Verification (Sets) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Sets

Purpose: Deduplicating 100,000 email recipients and checking blacklisted JWT `jti` IDs in `O(1)` average time.

What Topic Is Next → Next Topic 2.15: Set Operations

2.15 Set Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Union (`|` / `.union()`) • Intersection (`&` / `.intersection()`) • Difference (`-`) • Symmetric difference (`^`) • Production Pitfall Guard (Set Operations) • Hands-On Verification (Set Operations) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Set Operations

Purpose: Matching candidate skills against job required skills (`matched = candidate_skills & job_skills`, `missing = job_skills - candidate_skills`).

What Topic Is Next → Next Topic 2.16: Set Methods

2.16 Set Methods [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `add()` • `update()` • `discard()` (no error if missing) vs `remove()` (`KeyError` if missing) • `pop()` • Production Pitfall Guard (Set Methods) • Hands-On Verification (Set Methods) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Set Methods

Purpose: Safely revoking active WebSocket session IDs on disconnect using `active_Ids.discard(conn_id)` without `KeyError`.

What Topic Is Next → Next Topic 2.17: Dictionaries

2.17 Dictionaries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Insertion-ordered hash map (`dict` • Guaranteed since Python 3.7+) • Hashable keys • `O(1)` read/write/delete • Production Pitfall Guard (Dictionaries) • Hands-On Verification (Dictionaries) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Dictionaries

Purpose: Indexing employees/products by ID for `O(1)` lookup and representing JSON request/response payloads.

What Topic Is Next → Next Topic 2.18: Dictionary Methods

2.18 Dictionary Methods [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.keys()` • `.values()` • `.items()` dynamic views • `.get(k • Production Pitfall Guard (Dictionary Methods) • Hands-On Verification (Dictionary Methods) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Dictionary Methods

Purpose: Counting word frequencies via `counts[w] = counts.get(w, 0) + 1` and merging default config with overrides `defaults | overrides`.

What Topic Is Next → Module 2 Capstone (Week 2 Lab Build: Collections-Driven Academic, Cart & Word Analytics Pack) → Next Topic 3.1: Functions

Module 2 Hands-On Capstone Project (100 Marks): Week 2 Lab Build: Collections-Driven Academic, Cart & Word Analytics Pack — Build 6 collection-powered Python modules: Student Marks Manager, Employee Record Manager, Shopping Cart, Case-Insensitive Word Frequency Counter, Order-Preserving Deduplicator, and O(1) Dic...

MODULE 3 OF 7 • STAGE 1 • 11 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Functions, Type Hints, Modules, Packages & Exception Handling

Design clean, reusable, fault-tolerant Python modules: function parameters, mutable default pitfalls, `*args`/`**kwargs`, LEGB scope, lambdas, recursion, higher-order functions (`map`/`filter`/`reduce`), modules, packages, `if __name__ == "__main__":`, standard library (`os`, `sys`, `math`, `random`, `datetime`), `try`/`except`/`else`/`finally`, `raise`, custom exceptions, `pdb`, and `logging`.

Objectives: ✓ Write single-responsibility functions using positional, keyword, keyword-only, safe default (`None`), and `*args`/`**kwargs` parameters. • ✓ Apply LEGB variable scope rules, recursion with explicit base cases, and functional transformations cleanly. • ✓ Organize multi-file applications into Python modules and packages guarded by `if __name__ == "__main__":`. • ✓ Handle failures predictably with specific `try`/`except`/`else`/`finally` blocks, exception chaining (`raise ... from e`), and custom domain exceptions. • ✓ Debug state interactively via `breakpoint()`/`pdb` and replace raw `print()` calls with structured `logging` levels and handlers.

3.1 Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `def` declaration • Docstrings (PEP 257) • Single-responsibility design • Invocation • Production Pitfall Guard (Functions) • Hands-On Verification (Functions) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Functions

Purpose: Extracting tax, validation, and formatting logic into pure, unit-testable service functions.

What Topic Is Next → Next Topic 3.2: Function Parameters

3.2 Function Parameters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Positional arguments • Keyword arguments • Positional-only (`/`) • Keyword-only (`*`) parameter markers • Production Pitfall Guard (Function Parameters) • Hands-On Verification (Function Parameters) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Function Parameters

Purpose: Forcing boolean/option flags to be keyword-only (`def send_email(to, *, is_html=True):`) so callers never pass unreadable `send_email(x, True, False)`.

What Topic Is Next → Next Topic 3.3: Default Parameters

3.3 Default Parameters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Default parameter evaluation at `def` definition time (ONCE!) • The `None` sentinel pattern for mutable defaults • Production Pitfall Guard (Default Parameters) • Hands-On Verification (Default Parameters) • Default Parameters — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Default Parameters • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Safely initializing optional list/dict parameters via `tags: list[str] | None = None` so state never leaks across API requests!

What Topic Is Next → Next Topic 3.4: Variable Arguments

3.4 Variable Arguments [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Variadic positional `*args` (packed as `tuple`) • Variadic keyword `**kwargs` (packed as `dict`) • Call-site `*` / `**` unpacking • Production Pitfall Guard (Variable Arguments) • Hands-On Verification (Variable Arguments) • Variable Arguments — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Variable Arguments

Purpose: Forwarding `*args, **kwargs` cleanly inside decorators, logging wrappers, and Django CBV/serializer overrides.

What Topic Is Next → Next Topic 3.5: Return Statements

3.5 Return Statements [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Early-return guard clauses • Returning multiple values via tuple packing/unpacking • Consistent return types • Production Pitfall Guard (Return Statements) • Hands-On Verification (Return Statements) • Return Statements — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Return Statements

Purpose: Returning `(is_valid, errors)` tuples or domain objects consistently across all code paths.

What Topic Is Next → Next Topic 3.6: Variable Scope (LEGB)

3.6 Variable Scope (LEGB) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Local -> Enclosing -> Global -> Built-in (LEGB) name resolution • `global` vs `nonlocal` • Why mutable global state hurts concurrency • Production Pitfall Guard (Variable Scope (LEGB)) • Hands-On Verification (Variable Scope (LEGB)) • Variable Scope (LEGB) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Variable Scope (LEGB)

Purpose: Maintaining clean stateless functions while using `nonlocal` selectively in Stateful Closure factories.

What Topic Is Next → Next Topic 3.7: Lambda Functions

3.7 Lambda Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single-expression anonymous `lambda params: expr` functions for inline sort keys • Callbacks vs named `def` functions • Production Pitfall Guard (Lambda Functions) • Hands-On Verification (Lambda Functions) • Lambda Functions — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Lambda Functions • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Passing concise key extractors like `sorted(jobs, key=lambda j: j["salary_max"], reverse=True)`.

What Topic Is Next → Next Topic 3.8: Recursion

3.8 Recursion [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Base case + progress toward base case • Call stack frames • `sys.getrecursionlimit()` (~1000 in CPython • No tail-call optimization!) • Production Pitfall Guard (Recursion) • Hands-On Verification (Recursion) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Recursion

Purpose: Traversing nested category trees, JSON hierarchies, and directory structures cleanly.

What Topic Is Next → Next Topic 3.9: Higher-Order Functions

3.9 Higher-Order Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: First-class functions passed as arguments (strategy/validator callbacks) • Returned from factory functions • Production Pitfall Guard (Higher-Order Functions) • Hands-On Verification (Higher-Order Functions) • Higher-Order Functions — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Higher-Order Functions • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Injecting pluggable discount or tax calculation strategies into an order pricing pipeline.

What Topic Is Next → Next Topic 3.10: map() Function

3.10 map() Function [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lazy iterator `map(func • *iterables)` transformation vs list/generator comprehensions • Production Pitfall Guard (map() Function) • Hands-On Verification (map() Function) • map() Function — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for map() Function • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Parsing whitespace-separated integer streams lazily via `list(map(int, line.split()))`.

What Topic Is Next → Next Topic 3.11: filter() Function

3.11 filter() Function [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lazy predicate filtering `filter(pred • Iterable)` • `filter(None • Iterable)` truthy filtering • Production Pitfall Guard (filter() Function) • Hands-On Verification (filter() Function) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for filter() Function

Purpose: Streaming only valid, non-expired records through a memory-bounded pipeline.

What Topic Is Next → Next Topic 3.12: functools.reduce()

3.12 functools.reduce() [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `functools.reduce(fn • Iterable • Initializer)` cumulative folding vs Pythonic built-ins (`sum` • `min` • Production Pitfall Guard (functools.reduce()) • Hands-On Verification (functools.reduce()) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for functools.reduce()

Purpose: Composing a sequence of middleware/pipeline functions `reduce(lambda acc, fn: fn(acc), steps, data)`.

What Topic Is Next → Next Topic 3.13: Python Modules

3.13 Python Modules [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating reusable `.py` modules • Module namespace isolation • `sys.modules` import cache • Avoiding circular imports • Production Pitfall Guard (Python Modules) • Hands-On Verification (Python Modules) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Modules

Purpose: Extracting reusable financial/calculator formulas into `calculator.py` imported cleanly across CLI and test files.

What Topic Is Next → Next Topic 3.14: Python Packages

3.14 Python Packages [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-directory package hierarchy (`banking/models/` • `banking/services/`) • `__init__.py` • `__all__` • Production Pitfall Guard (Python Packages) • Hands-On Verification (Python Packages) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Packages

Purpose: Structuring Modular Banking, Django, and FastAPI backends into cohesive domain packages (`accounts`, `transactions`, `common`).

What Topic Is Next → Next Topic 3.15: Import System

3.15 Import System [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Absolute imports (`from app.services.auth import verify`) • Explicit relative imports • `sys.path` • Why `from mod import *` is banned • Production Pitfall Guard (Import System) • Hands-On Verification (Import System) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Import System

Purpose: Using explicit absolute imports (`from banking.services.ledger import transfer`) per PEP 8 so dependencies are crystal clear.

What Topic Is Next → Next Topic 3.16: Main Execution Guard (__name__)

3.16 Main Execution Guard (__name__) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Python sets `__name__ == "__main__"` on direct execution vs module path on `import` • Protecting CLI entrypoints • Production Pitfall Guard (Main Execution Guard (__name__)) • Hands-On Verification (Main Execution Guard (__name__)) • Main Execution Guard (__name__) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Main Execution Guard (__name__) • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Allowing `pytest` or another service to `import banking.cli` without accidentally launching an interactive `input()` prompt!

What Topic Is Next → Next Topic 3.17: Python Standard Library

3.17 Python Standard Library [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Practical backend usage of `os` • `sys` • `math` • `random`/`secrets` • Production Pitfall Guard (Python Standard Library) • Hands-On Verification (Python Standard Library) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Standard Library

Purpose: Generating UTC timestamps (`datetime.now(timezone.utc)`) and cryptographic OTP/reset tokens (`secrets.token_urlsafe(32)`).

What Topic Is Next → Next Topic 3.18: Exception Handling

3.18 Exception Handling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `try` • `except SpecificError as exc:` blocks • Exception hierarchy (`BaseException` vs `Exception`) • Preserving error context • Production Pitfall Guard (Exception Handling) • Hands-On Verification (Exception Handling) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Exception Handling

Purpose: Catching `ValueError` on malformed user inputs and returning friendly validation errors instead of crashing the process.

What Topic Is Next → Module 3 Capstone (Week 3 Lab Build: Modular Logged Banking & Calculator Package) → Next Topic 4.1: OOP Introduction

Module 3 Hands-On Capstone Project (100 Marks): Week 3 Lab Build: Modular Logged Banking & Calculator Package — Architect a multi-package Python Banking & Calculator application (`models`, `services`, `exceptions`, `cli`) featuring keyword-only parameters, custom `InsufficientFundsError` exceptions, `...

MODULE 4 OF 7 • STAGE 1 • 11 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Complete Python OOP (`@dataclass`, `ABC`, Dunders) & File/JSON I/O

Master every pillar and Python-specific mechanism of Object-Oriented Programming plus production file I/O: classes, objects, `__init__`, `self`, instance vs class variables, instance/class/static methods, encapsulation, single/multiple/multilevel/hierarchical inheritance, polymorphism & duck typing, abstraction & `abc`, composition, aggregation, MRO, magic/dunder methods, `@property`, `@dataclass`, text/binary file handling, `with` context managers, CSV, JSON, `pickle`, `pathlib`, and Mini-Project 1.

Objectives: ✓ Model clean domain entities using classes, `__init__`, instance vs class attributes, `@classmethod` factories, `@staticmethod`, `@property`, and `@dataclass`. • ✓ Apply encapsulation, single/multiple/multilevel/hierarchical inheritance, C3 Method Resolution Order (MRO), polymorphism, duck typing, and `abc.ABC` contracts. • ✓ Prefer composition and aggregation over brittle deep inheritance hierarchies and implement Pythonic dunder methods (`__repr__`, `__str__`, `__eq__`, `__len__`). • ✓ Read and persist UTF-8 text, CSV, and JSON files safely using `with` context managers and cross-platform `pathlib.Path` APIs. • ✓ Deliver Mini-Project 1: a complete OOP, file-backed, logged, exception-safe Python Employee Management System.

4.1 OOP Introduction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Procedural vs object-oriented design • Coupling data + behavior • The 4 OOP pillars in Python backend architecture • Production Pitfall Guard (OOP Introduction) • Hands-On Verification (OOP Introduction) • OOP Introduction — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for OOP Introduction

Purpose: Structuring Django models, DRF ViewSets, SQLAlchemy entities, and service classes cleanly.

What Topic Is Next → Next Topic 4.2: Classes

4.2 Classes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `class` statement • `CapWords` (`PascalCase`) PEP 8 naming • Class namespace dictionary (`__dict__`) • Docstrings • Production Pitfall Guard (Classes) • Hands-On Verification (Classes) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Classes

Purpose: Defining blueprints for `Employee`, `BankAccount`, and `JobPosting` domain models.

What Topic Is Next → Next Topic 4.3: Objects

4.3 Objects [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class instantiation `obj = ClassName(...)` • Unique object identity (`id(obj)`) • `isinstance()` • Attribute lookup order • Production Pitfall Guard (Objects) • Hands-On Verification (Objects) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Objects

Purpose: Instantiating independent request-scoped service and entity objects per user transaction.

What Topic Is Next → Next Topic 4.4: Constructors (__init__)

4.4 Constructors (__init__) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Object initialization via `__init__(self • ...)` (vs `__new__` allocator) • Validating invariants at construction time • Production Pitfall Guard (Constructors (__init__)) • Hands-On Verification (Constructors (__init__)) • Constructors (__init__) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Constructors (__init__)

Purpose: Guaranteeing an `Employee` object can NEVER exist in memory with a blank email or negative salary.

What Topic Is Next → Next Topic 4.5: self Instance Reference

4.5 self Instance Reference [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Python binds `instance.method(x)` to `Class.method(instance • X)` • Why explicit `self` is first parameter • Production Pitfall Guard (self Instance Reference) • Hands-On Verification (self Instance Reference) • self Instance Reference — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for self Instance Reference

Purpose: Accessing and updating per-instance state explicitly without ambiguous implicit `this` scoping.

What Topic Is Next → Next Topic 4.6: Instance Variables

4.6 Instance Variables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Object-specific attributes bound on `self.attr` inside `__init__` • Dynamic attribute storage in `obj.__dict__` • Production Pitfall Guard (Instance Variables) • Hands-On Verification (Instance Variables) • Instance Variables — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Instance Variables • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Storing isolated `emp_id`, `email`, `department`, and `salary` per `Employee` instance.

What Topic Is Next → Next Topic 4.7: Class Variables

4.7 Class Variables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class-level attributes shared across all instances (`company_name` • Constants) • Lookup fallback `obj.__dict__` -> `cls.__dict__` • Mutable class attribute danger • Production Pitfall Guard (Class Variables) • Hands-On Verification (Class Variables) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Class Variables

Purpose: Defining class-level constants (`MIN_BALANCE = 500`, Django model fields, Pydantic config) shared across all instances.

What Topic Is Next → Next Topic 4.8: Instance Methods

4.8 Instance Methods [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Behavior operating on `self` state • Mutator vs query methods • Calling sibling methods via `self.` • Production Pitfall Guard (Instance Methods) • Hands-On Verification (Instance Methods) • Instance Methods — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Instance Methods

Purpose: Implementing `account.deposit(amount)`, `account.withdraw(amount)`, and `employee.annual_ctc()` domain behaviors.

What Topic Is Next → Next Topic 4.9: Class Methods (@classmethod)

4.9 Class Methods (@classmethod) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@classmethod` decorator • `cls` class binding (subclass-aware!) • Named alternative constructors (`from_dict` • `from_csv_row`) • Production Pitfall Guard (Class Methods (@classmethod)) • Hands-On Verification (Class Methods (@classmethod)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Class Methods (@classmethod)

Purpose: Deserializing `Employee.from_dict(data)` and `Employee.from_csv_row(row)` cleanly when loading JSON/CSV files.

What Topic Is Next → Next Topic 4.10: Static Methods (@staticmethod)

4.10 Static Methods (@staticmethod) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@staticmethod` decorator for namespace-grouped pure helper functions requiring neither `self` nor `cls` • Production Pitfall Guard (Static Methods (@staticmethod)) • Hands-On Verification (Static Methods (@staticmethod)) • Static Methods (@staticmethod) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Static Methods (@staticmethod) • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy • Diagnostics, Profiling & Telemetry: tracemalloc, cProfile & mypy

Purpose: Grouping `Employee.is_valid_emp_code(code)` validation helpers alongside the domain class.

What Topic Is Next → Next Topic 4.11: Encapsulation

4.11 Encapsulation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Public attributes • `_protected` convention • `__private` name mangling (`_ClassName__attr`) • Invariant protection • Production Pitfall Guard (Encapsulation) • Hands-On Verification (Encapsulation) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Encapsulation

Purpose: Protecting `_balance` in `BankAccount` so balance changes ONLY occur through audited `deposit()` and `withdraw()` methods.

What Topic Is Next → Next Topic 4.12: Single Inheritance

4.12 Single Inheritance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Deriving specialized `class Sub(Base):` (`is-a` relationship) • `super().__init__(...)` • Extending base behavior • Production Pitfall Guard (Single Inheritance) • Hands-On Verification (Single Inheritance) • Single Inheritance — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Single Inheritance

Purpose: Inheriting `FullTimeEmployee(Employee)` and `ContractEmployee(Employee)` sharing core identity while specializing payout rules.

What Topic Is Next → Next Topic 4.13: Multiple Inheritance

4.13 Multiple Inheritance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inheriting from multiple bases `class View(AuthMixin • AuditMixin • BaseView):` • Cooperative `super()` mixin design • Production Pitfall Guard (Multiple Inheritance) • Hands-On Verification (Multiple Inheritance) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Multiple Inheritance

Purpose: Understanding exactly how Django REST Framework `GenericViewSet` combines `CreateModelMixin`, `ListModelMixin`, and `GenericAPIView`!

What Topic Is Next → Next Topic 4.14: Multilevel Inheritance

4.14 Multilevel Inheritance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Chained `GrandParent -> Parent -> Child` hierarchies (`Exception -> BankingError -> InsufficientFundsError`) • Keeping depth shallow • Production Pitfall Guard (Multilevel Inheritance) • Hands-On Verification (Multilevel Inheritance) • Multilevel Inheritance — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Multilevel Inheritance • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Structuring layered domain exception hierarchies (`AppError -> AuthError -> TokenExpiredError`) for granular or broad catching.

What Topic Is Next → Next Topic 4.15: Hierarchical Inheritance

4.15 Hierarchical Inheritance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multiple sibling subclasses (`SavingsAccount` • `CurrentAccount` • `SalaryAccount`) sharing one common parent (`Account`) • Production Pitfall Guard (Hierarchical Inheritance) • Hands-On Verification (Hierarchical Inheritance) • Hierarchical Inheritance — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Hierarchical Inheritance

Purpose: Modeling `EmailNotifier`, `SmsNotifier`, and `WebhookNotifier` subclasses sharing common `BaseNotifier` retry/logging.

What Topic Is Next → Next Topic 4.16: Polymorphism and Duck Typing

4.16 Polymorphism and Duck Typing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Method overriding across subtypes • Pythonic duck typing ("if it implements `.send()` • Treat it as a sender") • Production Pitfall Guard (Polymorphism and Duck Typing) • Hands-On Verification (Polymorphism and Duck Typing) • Polymorphism and Duck Typing — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Polymorphism

Purpose: Swapping `JsonEmployeeRepo` and `CsvEmployeeRepo` seamlessly behind the same `.save()` / `.list_all()` interface.

What Topic Is Next → Next Topic 4.17: Abstraction

4.17 Abstraction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Separating WHAT a repository/gateway contract promises from HOW PostgreSQL • Redis • Or JSON files implement it • Production Pitfall Guard (Abstraction) • Hands-On Verification (Abstraction) • Abstraction — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1)

dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Abstraction

Purpose: Keeping business services decoupled from storage details so Month 1 file storage upgrades cleanly to Month 3 SQLAlchemy.

What Topic Is Next → Next Topic 4.18: abc Module (Abstract Base Classes)

4.18 abc Module (Abstract Base Classes) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from abc import ABC • Abstractmethod` • Preventing instantiation of incomplete subclasses at instantiation time • Production Pitfall Guard (abc Module (Abstract Base Classes)) • Hands-On Verification (abc Module (Abstract Base Classes)) • abc Module (Abstract Base Classes) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for abc Module (Abstract Base Classes)

Purpose: Enforcing that every `PaymentGateway(ABC)` subclass implements `charge()` and `refund()` before instantiation.

What Topic Is Next → Module 4 Capstone (Project 1: Python Employee Management System) → Next Topic 5.1: Iterables

Module 4 Hands-On Capstone Project (100 Marks): Project 1: Python Employee Management System — Design and build a modular, object-oriented, file-backed Python 3.12 Employee Management Console Application supporting complete CRUD, search, JSON/CSV persistence, custom exception handling...

MODULE 5 OF 7 • STAGE 2 • 10 LIVE + LAB HOURS • 16 TOPICS (128 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Iterators, Generators (`yield`), Decorators & Context Managers

Elevate from core coder to idiomatic Python backend engineer: iterables, iterator protocol, `iter()`/`next()`, `yield` generators, generator expressions, decorators, nested decorators, closures, custom context managers, `contextlib`, comprehensions, unpacking, walrus `:=`, `match`/`case`, type hints, `typing`, generics, `Protocol`, advanced dataclasses, `Enum`, `collections`, `functools`, `itertools`, `re`, `os.environ`, `pyproject.toml`, `venv`, `pip`, and `Poetry`.

Objectives: ✓ Build lazy `O(1)`-memory streaming pipelines with custom iterators, `yield` generators, and generator expressions. • ✓ Author `functools.wraps`-preserving decorators, parameterized/nested decorators, closures, and `@contextmanager` utilities. • ✓ Write strictly typed Python 3.12 APIs using annotations, `TypeVar` generics, structural `Protocol`s, frozen/slotted `@dataclass`, and `Enum`s. • ✓ Solve real data-processing problems with `Counter`, `defaultdict`, `deque`, `lru_cache`, `itertools`, and compiled `re` regexes. • ✓ Configure reproducible projects with `venv`, `os.environ` 12-factor config, `pyproject.toml`, `pip`/`requirements.txt`, and `Poetry`.

5.1 Iterables [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Objects implementing `__iter__` (or sequence `__getitem__`) • Reusable multi-pass containers (`list` • `dict`) vs single-pass streams • Production Pitfall Guard (Iterables) • Hands-On Verification (Iterables) • Iterables — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Iterables

Purpose: Accepting general `Iterable[Employee]` parameters in service methods so callers can pass lists, tuples, sets, or DB cursors.

What Topic Is Next → Next Topic 5.2: Iterators

5.2 Iterators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stateful Iterator Protocol (`__iter__` returning `self` + `__next__` raising `StopIteration` when exhausted) • Production Pitfall Guard (Iterators) • Hands-On Verification (Iterators) • Iterators — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Iterators • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy • Diagnostics, Profiling & Telemetry: tracemalloc, cProfile & mypy

Purpose: Implementing paginated API cursor iterators that lazily fetch the next page only as consumers iterate.

What Topic Is Next → Next Topic 5.3: iter() and next()

5.3 iter() and next() [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Manual iteration via `it = iter(obj)` • `next(it • Default)` sentinel handling without `StopIteration` crashes • Production Pitfall Guard (iter() and next()) • Hands-On Verification (iter() and next()) • iter() and next() — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for iter()

Purpose: Extracting the CSV header row via `headers = next(reader, None)` before looping remaining data rows.

What Topic Is Next → Next Topic 5.4: Generators (yield)

5.4 Generators (yield) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generator functions pausing frame state at `yield` • `yield from` delegation • `O(1)` memory streaming • Production Pitfall Guard (Generators (yield)) • Hands-On Verification (Generators (yield)) • Generators (yield) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Generators (yield)

Purpose: Streaming 2-million-row CSV/DB exports in FastAPI `StreamingResponse` using `< 5 MB` RAM.

What Topic Is Next → Next Topic 5.5: Generator Expressions

5.5 Generator Expressions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lazy `(expr for x in iterable if cond)` parenthesized expressions passed directly into `sum()` • `any()` • `all()` • `max()` • Production Pitfall Guard (Generator Expressions) • Hands-On Verification (Generator Expressions) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Generator Expressions

Purpose: Checking `any(u.is_suspended for u in users)` with short-circuit termination and zero intermediate list allocation.

What Topic Is Next → Next Topic 5.6: Function Decorators

5.6 Function Decorators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@decorator` syntactic sugar (`fn = decorator(fn)`) • Preserving `__name__`/`__doc__` metadata with `@functools.wraps(fn)` • Production Pitfall Guard (Function Decorators) • Hands-On Verification (Function Decorators) • Function Decorators — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Function Decorators • Production Hardening &

Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Building `@timed_execution`, `@require_role`, and `@retry_on_transient_error` cross-cutting backend decorators.

What Topic Is Next → Next Topic 5.7: Nested Decorators

5.7 Nested Decorators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parameterized decorator factories (`@retry(max_attempts=3)`) • Bottom-up composition order of stacked `@d1` / `@d2` decorators • Production Pitfall Guard (Nested Decorators) • Hands-On Verification (Nested Decorators) • Nested Decorators — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Nested Decorators • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Stacking `@router.post("/jobs")` + `@rate_limit(10)` + `@audit_action("CREATE_JOB")` in predictable execution order.

What Topic Is Next → Next Topic 5.8: Closures

5.8 Closures [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lexical scope capture (`__closure__` cell objects) • Late-binding loop variable pitfalls in closures • Production Pitfall Guard (Closures) • Hands-On Verification (Closures) • Closures — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Closures • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Creating pre-configured validator and rate-limiter functions capturing threshold settings cleanly.

What Topic Is Next → Next Topic 5.9: Custom Context Managers (__enter__ and __exit__)

5.9 Custom Context Managers (__enter__ and __exit__) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Class-based context manager protocol (`__enter__` resource acquisition • `__exit__(exc_type • Exc_val • Tb)` cleanup/rollback) • Production Pitfall Guard (Custom Context Managers (__enter__ and __exit__)) • Hands-On Verification (Custom Context Managers (__enter__ and __exit__)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Custom Context Managers (__enter__ and __exit__)

Purpose: Implementing SQLAlchemy/psycopg transactional Unit-of-Work blocks that `commit()` on clean exit and `rollback()` on exception.

What Topic Is Next → Next Topic 5.10: contextlib (@contextmanager)

5.10 contextlib (@contextmanager) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from contextlib import contextmanager • Suppress • ExitStack` generator-based context managers (`try: yield resource finally: cleanup()`) • Production Pitfall Guard (contextlib (@contextmanager)) • Hands-On Verification (contextlib (@contextmanager)) • contextlib (@contextmanager) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for contextlib (@contextmanager)

Purpose: Writing FastAPI `lifespan` startup/shutdown hooks and request-scoped DB session generators cleanly.

What Topic Is Next → Next Topic 5.11: Advanced Comprehensions

5.11 Advanced Comprehensions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Set comprehensions `{...}` • Dict comprehensions `{k: v ...}` • Nested comprehensions with filtering • Production Pitfall Guard (Advanced Comprehensions) • Hands-On Verification (Advanced Comprehensions) • Advanced Comprehensions — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Advanced Comprehensions

Purpose: Extracting unique normalized skill tags `{tag.strip().lower() for job in jobs for tag in job.tags}`.

What Topic Is Next → Next Topic 5.12: Extended Iterable Unpacking

5.12 Extended Iterable Unpacking [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Starred target unpacking `first • *middle • Last = seq` • Nested tuple unpacking • Production Pitfall Guard (Extended Iterable Unpacking) • Hands-On Verification (Extended Iterable Unpacking) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Extended Iterable Unpacking

Purpose: Parsing `command, *args = line.split()` CLI tokens or `header, *rows = csv_lines` cleanly.

What Topic Is Next → Next Topic 5.13: Walrus Operator (:=)

5.13 Walrus Operator (:=) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Assignment expressions `if (m := pattern.search(line)):` • `while (chunk := f.read(8192)):` avoiding duplicate calls • Production Pitfall Guard (Walrus Operator (:=)) • Hands-On Verification (Walrus Operator (:=)) • Walrus Operator (:=) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Walrus Operator (:=) • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Reading fixed-size binary upload chunks (`while chunk := stream.read(65536):`) and capturing regex match objects inside `if` conditions.

What Topic Is Next → Next Topic 5.14: Structural Pattern Matching (match and case)

5.14 Structural Pattern Matching (match and case) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Python 3.10+ `match subject:` with sequence • Mapping (`case {"type": "paid" • "amount": amt}:`) • Class • Production Pitfall Guard (Structural Pattern Matching (match and case)) • Hands-On Verification (Structural Pattern Matching (match and case)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Structural Pattern Matching (match and case)

Purpose: Routing webhook event payloads (`payment.captured`, `payment.failed`, `refund.processed`) cleanly with structural destructuring.

What Topic Is Next → Next Topic 5.15: Type Hints and Annotations

5.15 Type Hints and Annotations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PEP 484/585/604 modern Python 3.12 syntax (`list[str]` • `dict[str • Int]` • `str | None`) • Production Pitfall Guard (Type Hints and Annotations) • Hands-On Verification (Type Hints and Annotations) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Type Hints

Purpose: Powering FastAPI automatic request parsing, OpenAPI generation, IDE autocomplete, and `mypy` static bug detection.

What Topic Is Next → Next Topic 5.16: typing Module (List, Dict, Optional, Union, Callable)

5.16 typing Module (List, Dict, Optional, Union, Callable) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Any` vs `object` • `Literal` • `TypedDict` • `Final` • Production Pitfall Guard (typing Module (List, Dict, Optional, Union, Callable)) • Hands-On Verification (typing Module (List, Dict, Optional, Union, Callable)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for typing Module (List, Dict, Optional, Union, Callable...

Purpose: Typing structured JSON payloads with `TypedDict` and constrained status strings with `Literal["draft", "published", "closed"]`.

What Topic Is Next → Module 5 Capstone (Week 5 Lab Build: Streaming Log Analyzer & Typed Config Engine) → Next Topic 6.1: Python Execution Model

Module 5 Hands-On Capstone Project (100 Marks): Week 5 Lab Build: Streaming Log Analyzer & Typed Config Engine — Create an O(1)-memory generator log-analytics pipeline using `@functools.wraps` timing/retry decorators, custom `@contextmanager` timers, `StrEnum` levels, frozen `@dataclass(slots=True)` DT...

MODULE 6 OF 7 • STAGE 2 • 10 LIVE + LAB HOURS • 16 TOPICS (128 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Python Memory Model, GIL, Multiprocessing & `asyncio` Concurrency

Understand CPython under the hood and choose the right concurrency model every time: interpreter execution, CPython internals, bytecode, memory allocation, reference counting, cyclic GC, mutable references, copying, the Global Interpreter Lock (GIL), `threading`, `Lock`/`RLock`, race conditions, deadlocks, `ThreadPoolExecutor`, `multiprocessing`, `ProcessPoolExecutor`, IPC `Queue`/`Pipe`, `async`/`await`, event loop, `asyncio`, `create_task`, `gather`, `async with`, async iterators/generators, `httpx`, and concurrency model selection.

Objectives: ✓ Explain CPython bytecode execution, `PyObject` reference counting (`sys.getrefcount`), cyclic `gc` generations, and the GIL. • ✓ Execute concurrent I/O safely with `threading`, `Lock`/`RLock`, and `ThreadPoolExecutor` while preventing race conditions and deadlocks. • ✓ Bypass the GIL for CPU-bound workloads using `multiprocessing`, `ProcessPoolExecutor`, and `Queue`/`Pipe` IPC. • ✓ Write non-blocking `asyncio` coroutines using `async`/`await`, `create_task`, `gather`, `async with`, async generators, and `httpx.AsyncClient`. • ✓ Select accurately between `asyncio` (high-concurrency I/O), `ThreadPoolExecutor` (blocking I/O libraries), and `ProcessPoolExecutor` (CPU-bound work).

6.1 Python Execution Model [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Source parsing -> AST -> code objects -> bytecode evaluation loop on the Python Virtual Machine stack frame • Production Pitfall Guard (Python Execution Model) • Hands-On Verification (Python Execution Model) • Python Execution Model — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Execution Model • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy • Diagnostics, Profiling & Telemetry: tracemalloc, cProfile & mypy

Purpose: Understanding import-time execution vs call-time execution in Django models and FastAPI route registration.

What Topic Is Next → Next Topic 6.2: CPython Runtime Architecture

6.2 CPython Runtime Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: CPython reference interpreter vs PyPy/Cython • C extension ABI • Why NumPy/psycopg release the GIL in C code • Production Pitfall Guard (CPython Runtime Architecture) • Hands-On Verification (CPython Runtime Architecture) • CPython Runtime Architecture — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for CPython Runtime Architecture

Purpose: Knowing why C-backed libraries (`orjson`, `pydantic-core` in Rust, `psycopg`) achieve native speed inside CPython.

What Topic Is Next → Next Topic 6.3: Python Bytecode Compilation (dis)

6.3 Python Bytecode Compilation (dis) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inspecting stack-machine opcodes (`LOAD_FAST` • `BINARY_OP` • `RETURN_VALUE`) • `.pyc` caching • Production Pitfall Guard (Python Bytecode Compilation (dis)) • Hands-On Verification (Python Bytecode Compilation (dis)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Bytecode Compilation (dis)

Purpose: Understanding why local variable lookups (`LOAD_FAST`) are faster than global lookups (`LOAD_GLOBAL`) in hot loops.

What Topic Is Next → Next Topic 6.4: Memory Allocation in Python

6.4 Memory Allocation in Python [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Private CPython heap • `PyObject` header (`ob_refcnt` + `ob_type`) • `pymalloc` small-object arenas • `tracemalloc` profiling • Production Pitfall Guard (Memory Allocation in Python) • Hands-On Verification (Memory Allocation in Python) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Memory Allocation in Python

Purpose: Profiling memory spikes in Celery workers using `tracemalloc` and reducing per-object overhead with `__slots__`.

What Topic Is Next → Next Topic 6.5: Reference Counting

6.5 Reference Counting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immediate deterministic deallocation when `ob_refcnt` drops to `0` • Inspecting `sys.getrefcount()` • Strong vs `weakref` references • Production Pitfall Guard (Reference Counting) • Hands-On Verification (Reference Counting) • Reference Counting — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Reference Counting

Purpose: Understanding why file handles in `with` blocks and unreferenced local objects free immediately when scope exits.

What Topic Is Next → Next Topic 6.6: Cyclic Garbage Collection (gc)

6.6 Cyclic Garbage Collection (gc) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why reference cycles (`a.peer = b` • `B.peer = a`) never hit refcount `0` • Generational `gc` (Gen 0/1/2) • `gc.collect()` • Production Pitfall Guard (Cyclic Garbage Collection (gc)) • Hands-On Verification (Cyclic Garbage Collection (gc)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Cyclic Garbage Collection (gc)

Purpose: Avoiding accidental circular object graphs in custom caches and long-lived worker processes.

What Topic Is Next → Next Topic 6.7: Mutable Object References in Memory

6.7 Mutable Object References in Memory [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Shared references across containers • In-place mutation (`+=` on `list` calls `__iadd__`) vs rebinding (`+=` on `tuple`/`str` creates new object) • Production Pitfall Guard (Mutable Object References in Memory) • Hands-On Verification (Mutable Object References in Memory) • Mutable Object References in Memory — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Mutable Object References in Memory • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Preventing shared default config dictionaries from being mutated across concurrent requests.

What Topic Is Next → Next Topic 6.8: Shallow vs Deep Copying in Memory

6.8 Shallow vs Deep Copying in Memory [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Memory graph topology of alias (`=`) • Shallow copy (`copy.copy`) • Deep copy (`copy.deepcopy`) on nested state • Production Pitfall Guard (Shallow vs Deep Copying in Memory) • Hands-On Verification (Shallow vs Deep Copying in Memory) • Shallow vs Deep Copying in Memory — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Shallow vs Deep Copying in Memory

Purpose: Ensuring isolated request context copies without duplicating shared read-only lookup tables.

What Topic Is Next → Next Topic 6.9: Global Interpreter Lock (GIL)

6.9 Global Interpreter Lock (GIL) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: What the CPython GIL protects • Why CPU-bound Python bytecode runs 1 thread at a time per process • Why I/O releases the GIL • Production Pitfall Guard (Global Interpreter Lock (GIL)) • Hands-On Verification (Global Interpreter Lock (GIL)) • Global Interpreter Lock (GIL) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Global Interpreter Lock (GIL)

Purpose: Explaining clearly in interviews why Gunicorn spawns multiple worker PROCESSES (`-w 4`) while each Uvicorn worker handles thousands of async I/O sockets.

What Topic Is Next → Next Topic 6.10: Threading Fundamentals

6.10 Threading Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: OS threads inside a single shared-memory process • Main thread vs daemon threads • Preemptive context switching • Production Pitfall Guard (Threading Fundamentals) • Hands-On Verification (Threading Fundamentals) • Threading Fundamentals — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Threading Fundamentals

Purpose: Offloading blocking SMTP/S3 SDK calls onto background worker threads without blocking the main loop.

What Topic Is Next → Next Topic 6.11: Creating and Managing Threads

6.11 Creating and Managing Threads [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `threading.Thread(target=... • Args=...)` • `.start()` • `.join(timeout=...)` • Production Pitfall Guard (Creating and Managing Threads) • Hands-On Verification (Creating and Managing Threads) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Creating

Purpose: Running parallel health checks across 5 downstream endpoints and joining all results with explicit timeouts.

What Topic Is Next → Next Topic 6.12: Thread Synchronization (Lock and RLock)

6.12 Thread Synchronization (Lock and RLock) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mutual exclusion with `threading.Lock()` (`with lock:`) • Re-entrant `RLock()` when a locked method calls another locked method on `self` • Production Pitfall Guard (Thread Synchronization (Lock and RLock)) • Hands-On Verification (Thread Synchronization (Lock and RLock)) • Thread Synchronization (Lock and RLock) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Thread Synchronization (Lock and RLock) • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Protecting an in-memory token bucket rate limiter accessed concurrently by Gunicorn gthread workers.

What Topic Is Next → Next Topic 6.13: Race Conditions

6.13 Race Conditions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Check-then-act (`if balance >= amt: balance -= amt`) • Read-modify-write (`counter += 1` is multiple bytecodes!) non-atomicity • Production Pitfall Guard (Race Conditions) • Hands-On Verification (Race Conditions) • Race Conditions — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Race Conditions • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Preventing double-withdrawal or overselling the last inventory item under concurrent requests.

What Topic Is Next → Next Topic 6.14: Deadlocks and Prevention

6.14 Deadlocks and Prevention [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Circular lock wait (`Thread 1 locks A -> waits B` • `Thread 2 locks B -> waits A`) • Prevention via canonical lock ordering (`id` sort) & timeouts • Production Pitfall Guard (Deadlocks and Prevention) • Hands-On Verification (Deadlocks and Prevention) • Deadlocks and Prevention — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Deadlocks

Purpose: Ordering account IDs `first, second = sorted([acc_a, acc_b], key=lambda a: a.id)` in bank transfers so deadlocks are mathematically impossible!

What Topic Is Next → Next Topic 6.15: Thread Pools (ThreadPoolExecutor)

6.15 Thread Pools (ThreadPoolExecutor) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from concurrent.futures import ThreadPoolExecutor • As_completed` • `executor.submit()` • `executor.map()` • Production Pitfall Guard (Thread Pools (ThreadPoolExecutor)) • Hands-On Verification (Thread Pools (ThreadPoolExecutor)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Thread Pools (ThreadPoolExecutor)

Purpose: Fetching 20 S3 object metadata headers concurrently with bounded `max_workers=8` instead of spawning unbounded raw threads.

What Topic Is Next → Next Topic 6.16: Multiprocessing (Process-Based Parallelism)

6.16 Multiprocessing (Process-Based Parallelism) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Separate OS processes • Separate Python heaps + separate GILs for true multi-core CPU parallelism • `spawn` vs `fork` start methods • Production Pitfall Guard (Multiprocessing (Process-Based Parallelism)) • Hands-On Verification (Multiprocessing (Process-Based Parallelism)) • Multiprocessing (Process-Based Parallelism) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Multiprocessing (Process-Based Parallelism)

Purpose: Parallelizing CPU-heavy PDF report generation, image resizing, or cryptographic hashing across all CPU cores.

What Topic Is Next → Module 6 Capstone (Week 6 Lab Build: Concurrency Benchmark & Async HTTP Aggregator) → Next Topic 7.1: Software Testing Fundamentals

Module 6 Hands-On Capstone Project (100 Marks): Week 6 Lab Build: Concurrency Benchmark & Async HTTP Aggregator — Benchmark CPU-bound prime counting (`ProcessPoolExecutor`) vs blocking I/O (`ThreadPoolExecutor` + deadlock-free `Lock` ordering) vs non-blocking `asyncio.TaskGroup` + `Semaphore`-throttled...

MODULE 7 OF 7 • STAGE 2 • 11 LIVE + LAB HOURS • 14 TOPICS (112 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Automated Unit Testing (`pytest`), Packaging, Git Workflow & Python Capstone

Verify Python code with `pytest` fixtures/parametrization, manage `pyproject.toml` virtual environments, enforce clean Git PR habits, and deliver the Python 3.12 Capstone.

Objectives: ✓ Verify Python code with `pytest` fixtures/parametrization, manage `pyproject.toml` virtual environments, enforce clean Git PR habits, and deliver the Python 3.12 Capstone. • ✓ Implement, test, and verify production-grade code and edge-case guardrails across all topics in Week 7.

7.1 Software Testing Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Arrange-Act-Assert (AAA) structure • Deterministic repeatability (no sleep/network/clock flakiness!) • Catching regressions in CI before deploy • Production Pitfall Guard (Software Testing Fundamentals) • Hands-On Verification (Software Testing Fundamentals) • Software Testing Fundamentals — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Software Testing Fundamentals

Purpose: Refactoring SQLAlchemy queries or pricing formulas confidently knowing 150 automated tests verify every invariant in 4 seconds.

What Topic Is Next → Next Topic 7.2: Testing Pyramid (Unit, Integration, Functional)

7.2 Testing Pyramid (Unit, Integration, Functional) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fast pure-Python Unit tests (70%) + real-DB/API Integration tests (25%) + end-to-end smoke workflows (5%) • Production Pitfall Guard (Testing Pyramid (Unit, Integration, Functional)) • Hands-On Verification (Testing Pyramid (Unit, Integration, Functional)) • Testing Pyramid (Unit, Integration, Functional) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Testing Pyramid (Unit, Integration, Functional) • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy • Diagnostics, Profiling & Telemetry: tracemalloc, cProfile & mypy

Purpose: Testing 30 tax/coupon edge cases in `15ms` as pure unit tests while testing SQL constraints & HTTP auth in integration tests.

What Topic Is Next → Next Topic 7.3: Test-Driven Development (TDD) Fundamentals

7.3 Test-Driven Development (TDD) Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Red (write failing test specifying requirement) -> Green (minimal passing implementation) -> Refactor (clean up while keeping tests green) • Production Pitfall Guard (Test-Driven Development (TDD) Fundamentals) • Hands-On Verification (Test-Driven Development (TDD) Fundamentals) • Test-Driven Development (TDD) Fundamentals — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Test-Driven Development (TDD) Fundamentals • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy • Diagnostics, Profiling & Telemetry: tracemalloc, cProfile & mypy

Purpose: Reproducing a reported production bug FIRST as a failing `pytest` test so that exact bug can never regress again!

What Topic Is Next → Next Topic 7.4: Python Built-In unittest Framework

7.4 Python Built-In unittest Framework [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `unittest.TestCase` • `setUp`/`tearDown` • `self.assertEqual`/`assertRaises` • Why Django's `django.test.TestCase` subclasses `unittest.TestCase` • Production Pitfall Guard (Python Built-In unittest Framework) • Hands-On Verification (Python Built-In unittest Framework) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Built-In unittest Framework

Purpose: Maintaining standard `unittest` suites and running them seamlessly under both `python -m unittest` and `pytest`.

What Topic Is Next → Next Topic 7.5: Pytest Setup, Configuration and Execution

7.5 Pytest Setup, Configuration and Execution [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `pytest` + `pytest-asyncio` + `pytest-cov` • Configuring `[tool.pytest.ini_options]` in `pyproject.toml` • `-k` filter • `-x` stop-on-first-fail • Production Pitfall Guard (Pytest Setup, Configuration and Execution) • Hands-On Verification (Pytest Setup, Configuration and Execution) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Pytest Setup

Purpose: Running `pytest -k "auth and not slow" -x` for sub-second feedback loops during active coding.

What Topic Is Next → Next Topic 7.6: Writing Expressive Pytest Test Functions

7.6 Writing Expressive Pytest Test Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Descriptive `test_<unit>_<scenario>_<expected>()` naming (`test_apply_job_when_already_applied_returns_409`) • Plain `assert` introspection • Production Pitfall Guard (Writing Expressive Pytest Test Functions) • Hands-On Verification (Writing Expressive Pytest Test Functions) • Writing Expressive Pytest Test Functions — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12

Bytecode & Heap Execution & Memory Lifecycle for Writing Expressive Pytest Test Functions • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: Knowing immediately from CI failure output `FAILED test_update_job_by_other_employer_returns_403` what business invariant broke.

What Topic Is Next → Next Topic 7.7: Assertions and Exception Checks (pytest.raises)

7.7 Assertions and Exception Checks (pytest.raises) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Rich `pytest` diffing on dicts/lists • `pytest.approx()` for floats • `with pytest.raises(InsufficientFundsError • Match="Balance"):` • Production Pitfall Guard (Assertions and Exception Checks (pytest.raises)) • Hands-On Verification (Assertions and Exception Checks (pytest.raises)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Assertions

Purpose: Asserting BOTH exact exception class AND `match=` message regex so the right validation failed for the right reason!

What Topic Is Next → Next Topic 7.8: Reusable Test State with Pytest Fixtures

7.8 Reusable Test State with Pytest Fixtures [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@pytest.fixture(scope="function" | "module" | "session")` in `conftest.py` • Composable dependency injection • `yield` teardown • `tmp_path` built-in fixture • Production Pitfall Guard (Reusable Test State with Pytest Fixtures) • Hands-On Verification (Reusable Test State with Pytest Fixtures) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Reusable Test State with Pytest Fixtures

Purpose: Creating `db_session`, `auth_client`, `sample_employer`, and `tmp_path` upload fixtures shared cleanly across `tests/`.

What Topic Is Next → Next Topic 7.9: Version Control Fundamentals

7.9 Version Control Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Distributed Version Control (DVCS) • Auditability • Blame history • Safe experimentation • Production Pitfall Guard (Version Control Fundamentals) • Hands-On Verification (Version Control Fundamentals) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Version Control Fundamentals

Purpose: Tracing exact commit/author (`git blame` / `git bisect`) when a production API regression appears.

What Topic Is Next → Next Topic 7.10: Git Architecture (Working Directory, Staging, Repository)

7.10 Git Architecture (Working Directory, Staging, Repository) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Three-tree model: Working Tree -> Staging Area (`.git/index`) -> Local `.git/objects` DAG (`blob` • `tree` • `commit` SHA-1/SHA-256) • Production Pitfall Guard (Git Architecture (Working Directory, Staging, Repository)) • Hands-On Verification (Git Architecture (Working Directory, Staging, Repository)) • Git Architecture (Working Directory, Staging, Repository) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Git Architecture (Working Directory, Staging, Reposi...

Purpose: Staging only clean bugfix hunks into `.git/index` while keeping local scratch experiments unstaged.

What Topic Is Next → Next Topic 7.11: Git Installation and Configuration

7.11 Git Installation and Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git config --global user.name` • `user.email` • `init.defaultBranch main` • `core.autocrlf` (Windows `true` vs Linux `input`) • Production Pitfall Guard (Git Installation and Configuration) • Hands-On Verification (Git Installation and Configuration) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Git Installation

Purpose: Preventing CRLF line-ending noise when Windows developers push Python/bash scripts deployed to Linux Docker containers.

What Topic Is Next → Next Topic 7.12: git init (Repository Initialization)

7.12 git init (Repository Initialization) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Initializing `.git` at project root • Verifying repository boundaries • Immediate `.gitignore` creation before first `git add` • Production Pitfall Guard (git init (Repository Initialization)) • Hands-On Verification (git init (Repository Initialization)) • git init (Repository Initialization) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for git init (Repository Initialization)

Purpose: Bootstrapping clean standalone microservice repositories with `.gitignore` and `README.md` from commit #1.

What Topic Is Next → Next Topic 7.13: git status (Inspecting Working Tree)

7.13 git status (Inspecting Working Tree) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Interpreting untracked • Unstaged • Staged • Ahead/behind remote tracking indicators before every commit • Production Pitfall Guard (git status (Inspecting Working Tree)) • Hands-On Verification (git status (Inspecting Working Tree)) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for git status (Inspecting Working Tree)

Purpose: Catching accidental `.env` or `dump.sql` files BEFORE staging changes.

What Topic Is Next → Next Topic 7.14: git add (Staging Changes)

7.14 git add (Staging Changes) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit file staging `git add src/models.py` • Interactive hunk staging `git add -p` • Unstaging via `git restore --staged` • Production Pitfall Guard (git add (Staging Changes)) • Hands-On Verification (git add (Staging Changes)) • git add (Staging Changes) — Interview Readiness • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for git add (Staging Changes)

Purpose: Splitting two unrelated edits in the same file into two clean, reviewable commits using `git add -p`.

What Topic Is Next → Module 7 Capstone → End-of-Course Live Client Project: Core & Advanced Python 3.12 Programming — Enterprise Client Production Platform

Module 7 Hands-On Capstone Project (100 Marks): Capstone: Modular Python 3.12 Banking, Inventory & Async Log-Analyzer CLI Suite — Build a multi-package Python 3.12 application combining an OOP Banking & Inventory Ledger (JSON/CSV persistence, custom exceptions, @dataclass models) with an

AsyncIO Concurrent Log-Analyzer...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

End-of-Course Live Client Production Project + Official Internship Confirmation Letter

Project: Core & Advanced Python 3.12 Programming — Enterprise Client Production Platform • **Confirmed Role:** Core & Advanced Python 3.12 Programming Engineering Project Intern

Domain & Production Infra: Live Enterprise Programming & Backend Client Engagement (Python 3.12, Lists, Dicts & Sets, Functions & Type Hints, OOP & @dataclass, Decorators & Generators • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular Python 3.12, Lists, Dicts & Sets, Functions & Type Hints, OOP & @dataclass, Decorators & Generators production architecture.
- Implemented core application modules, input validation, security hardening, and optimized data/state workflows.
- Completed peer Git pull-request code reviews, automated verification suites, and edge-case debugging sprints.
- Deployed the verified build to live client production with monitoring, documentation, and mentor sign-off.

Target Job Roles: Core & Advanced Python 3.12 Programming Developer, Associate Programming & Backend Engineer, Software Development Engineer (SDE-1)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 LPA – ₹8.5 LPA **Mid-Level Product Engineer (1–3 Yrs):** ₹9.5 LPA – ₹19.0 LPA **Senior / Multi-Stack Specialist (3–6+ Yrs):** ₹20.0 LPA – ₹38.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 95400 07917 • Web: https://mockattempt.com/courses/python-programming-training

Try 2 Free Topic-Wise Timed Topic Mock Tests & Spoken MockAttempt AI Interviews daily on every topic above before enrolling.