Admissions & Support:
support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: PYTH • **Updated:** 03 Nov 2026
https://mockattempt.com/courses/python-training

PYTH • PROGRAMMING & BACKEND **BEGINNER TO JOB-READY PYTHON DEVELOPER**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

Python Training

Master Python from scratch to OOP, DSA, Decorators, Regex, Web Scraping, SQLite/MySQL & Tkinter GUI — with runnable programs, 376 Topic Mock Tests (17,360 MCQs), and Live Voice + MockAttempt AI Technical Interviews at the most affordable fee.

Duration: 16–18 Weeks • Live + Hands-On Projects (205 Hrs) • **Modules:** 16 • **Live Topics:** 52 • **Subtopics:** 416+ • **Labs & Tasks:** 104 • **Mocks / AI Vivas:** 376 / 376 • **Projects:** 16 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹2,499 (Net Tuition ₹2,118 + Included 18% GST ₹381) • **Split Option:** 100% Credit to Job Track • **Effective ROI:** ~₹12/Guided Hr (~₹48/Live Topic)

Faculty: 100% Active Working IT Professionals • **Live Batch Tracks (7:00 AM – 8:00 PM IST Flexible Slots):** Weekday Live Online Batch (Mon – Fri • 1 Hr/Day • 7:00 AM – 8:00 PM) (Monday to Friday (1 Hour Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 1-Hour Slot Based on Trainer Availability) • 17–19 Weeks Fast-Track) | Weekend Live Online Batch (Sat – Sun • 2 Hrs/Day • 7:00 AM – 8:00 PM) (Saturday & Sunday (2 Hours Live Session / Day) • 07:00 AM – 08:00 PM IST (Students Can Choose Any 2-Hour Slot Based on Trainer Availability) • 19–21 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 • Data Structures & OOP • File/JSON/CSV ETL • Algorithms • Regex & Scraping • MySQL & SQLite3 • Tkinter GUI

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Write clean, PEP-8 compliant Python 3 programs from fundamentals to advanced OOP, Decorators & Generators
2. Build 16 portfolio-ready Python Capstone Projects across CLI automation, ETL pipelines, Web Scraping, Databases, and Tkinter Desktop GUI
3. Practice 376 Topic-Wise Timed Mock Exams containing 17,360 curated Python MCQs with explanations
4. Clear 376 Spoken Voice + MockAttempt AI Technical Interviews with real-time Speech-to-Text verification
5. Python 3.12+ static type hints, dataclasses, Enums, and modular package layouts (`src/` layout with `pyproject.toml`).
6. Crack LeetCode and HackerRank problems in Python 3.12 with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
7. Memory-safe iterators, generators, and context managers (`with`) for large file, stream, and database workloads.
8. Structured exception hierarchies, environment-driven configs, and structured JSON logging instead of `print()` debugging.

Eligibility & Prerequisites: Zero prior programming experience required — starts from installing Python & writing your first program • 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 live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews. • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • 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 (16 MODULES • 52 LIVE TOPICS • 416+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Stage 1: Day-1 Foundations & Core Mental Models (15 Topics / 120 Subtopics) • Modules 5–8: Stage 2: Applied Engineering, Data & API Workflows (13 Topics / 104 Subtopics) • Modules 9–12: Stage 3: Internals, Concurrency, Security & Performance (12 Topics / 96 Subtopics) • Modules 13–16: Stage 4: Production Architecture, Testing, CI/CD & Expert Mastery (12 Topics / 96 Subtopics)

MODULE 1 OF 16 • STAGE 1 • 10 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Python Fundamentals, Setup, Syntax & Operators

Get started with CPython architecture, interactive vs script execution, indentation rules, keywords, and all 7 Python operator families.

Objectives: ✓ What is Python — Architecture, Features & Applications — History of Python, CPython bytecode execution, dynamic typing, standard library ecosystem, and real-world applications. • ✓ Python Syntax, Indentation, Identifiers & Comments — Interactive REPL vs .py script mode, 4-space indentation blocks, valid identifiers, reserved keywords, and docstrings. • ✓ Python Operators & Precedence (Arithmetic, Bitwise, Logical, Identity) — Deep dive into //, **, %, comparison chaining, short-circuit and/or, bitwise shifts, in/not in, and is vs ==. • ✓ Build, verify, and ship Module Capstone: Smart CLI Personal Expense, GST & Loan EMI Calculator.

1.1 What is Python — Architecture, Features & Applications

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Interpreted & Bytecode • Core Syntax, CLI & Production APIs: print(), type(), sys.version • Deep-Dive Mechanics: Dynamic typing & Object-oriented • Advanced Patterns: CPython & Open-source • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Python Environment & Object Identity Inspector • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for What is Python

Purpose: To understand and apply What is Python — Architecture, Features & Applications in real Python systems — specifically: History of Python, CPython bytecode execution, dynam...

What Topic Is Next → Next Topic 1.2: Python Syntax, Indentation, Identifiers & Comments

1.2 Python Syntax, Indentation, Identifiers & Comments

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Indentation & Identifier • Core Syntax, CLI & Production APIs: 4-space indentation, # comments, """docstrings""", keyword.kwlist • Deep-Dive Mechanics: Keyword & Docstring • Advanced Patterns: Script mode & Snake_case • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: PEP-8 Indentation, Docstrings & Multi-Line Calculation • 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: To understand and apply Python Syntax, Indentation, Identifiers & Comments in real Python systems — specifically: Interactive REPL vs .py script mode, 4-space indentation...

What Topic Is Next → Next Topic 1.3: Python Operators & Precedence (Arithmetic, Bitwise, Logical, Identity)

1.3 Python Operators & Precedence (Arithmetic, Bitwise, Logical, Identity)

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Floor division & Exponent • Core Syntax, CLI & Production APIs: +, -, *, /, //, %, **, ==, is, in, &, |, ^ • Deep-Dive Mechanics: Short-circuit & Bitwise • Advanced Patterns: Identity is & Precedence • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: All-in-One Operator & Identity vs Equality Verifier • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Operators

Purpose: To understand and apply Python Operators & Precedence (Arithmetic, Bitwise, Logical, Identity) in real Python systems — specifically: Deep dive into //, **, %, comparison...

What Topic Is Next → Module 1 Capstone (Smart CLI Personal Expense, GST & Loan EMI Calculator) → Next Topic 2.1: Python Variables, Multiple Unpacking & Scope Rules

Module 1 Hands-On Capstone Project (100 Marks): Smart CLI Personal Expense, GST & Loan EMI Calculator — Build an interactive command-line financial toolkit in Python 3 that computes GST tax splits (CGST/SGST), monthly loan EMIs using exponentiation operators, and budget alerts.

MODULE 2 OF 16 • STAGE 1 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Variables, Data Types, Type Casting & String Manipulation

Master variable unpacking, LEGB scope, built-in numeric/sequence/mapping types, explicit type conversion, and string slicing & methods.

Objectives: ✓ Python Variables, Multiple Unpacking & Scope Rules — Object references, id(), multiple assignment, tuple unpacking, swapping variables, and local vs global scope. • ✓ Python Built-In Data Types (Numeric, Sequence, Set, Mapping, Boolean) — Complete tour of int, float, complex, str, list, tuple, range, dict, set, frozenset, bool, bytes, and NoneType. • ✓ Implicit & Explicit Type Casting in Python — Automatic numeric widening vs explicit conversion constructors int(), float(), str(), list(), tuple(), set(), and bool() truthiness. • ✓ Python Strings — Indexing, Slicing, Formatting & Built-In Methods — Unicode immutability, negative indexing, step slicing [::-1], f-strings, split(), join(), strip(), replace(), and find(). • ✓ Build, verify, and ship Module Capstone: Automated Resume & Invoice TextSanitizer CLI.

2.1 Python Variables, Multiple Unpacking & Scope Rules

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Reference & Unpacking • Core Syntax, CLI & Production APIs: x, y = y, x; a, *rest = seq; id(); global / nonlocal • Deep-Dive Mechanics: Local scope & Global • Advanced Patterns: Id() & Dynamic binding • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Tuple Unpacking, Star Assignment & Scope Demo • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Variables

Purpose: To understand and apply Python Variables, Multiple Unpacking & Scope Rules in real Python systems — specifically: Object references, id(), multiple assignment, tuple unpa...

What Topic Is Next → Next Topic 2.2: Python Built-In Data Types (Numeric, Sequence, Set, Mapping, Boolean)

2.2 Python Built-In Data Types (Numeric, Sequence, Set, Mapping, Boolean)

[50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Int & Float • Core Syntax, CLI & Production APIs: isinstance(obj, cls), type(), int, float, complex, range, bytes • Deep-Dive Mechanics: Complex & Bool • Advanced Patterns: NoneType & Isinstance • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Built-In Data Types & Decimal Precision Showcase • 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 Data Types (Numeric, Sequence, Set,...

Purpose: To understand and apply Python Built-In Data Types (Numeric, Sequence, Set, Mapping, Boolean) in real Python systems — specifically: Complete tour of int, float, complex,...

What Topic Is Next → Next Topic 2.3: Implicit & Explicit Type Casting in Python

2.3 Implicit & Explicit Type Casting in Python [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Implicit casting & Explicit casting • Core Syntax, CLI & Production APIs: int(x, base), float(x), str(x), bool(x), list(iterable) • Deep-Dive Mechanics: Int() & Float() • Advanced Patterns: Truthiness & Base conversion • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe Numeric Base & Explicit Type Conversion Utility • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Implicit

Purpose: To understand and apply Implicit & Explicit Type Casting in Python in real Python systems — specifically: Automatic numeric widening vs explicit conversion constructors i...

What Topic Is Next → Next Topic 2.4: Python Strings — Indexing, Slicing, Formatting & Built-In Methods

2.4 Python Strings — Indexing, Slicing, Formatting & Built-In Methods [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Slicing & Immutability • Core Syntax, CLI & Production APIs: s[start:stop:step], f"{val:.2f}", .split(), .join(), .strip(), .replace() • Deep-Dive Mechanics: F-string & Split and join • Advanced Patterns: Negative index & Strip • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: String Slicing, Palindrome Check & Slug Formatter • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Strings

Purpose: To understand and apply Python Strings — Indexing, Slicing, Formatting & Built-In Methods in real Python systems — specifically: Unicode immutability, negative indexing,...

What Topic Is Next → Module 2 Capstone (Automated Resume & Invoice TextSanitizer CLI) → Next Topic 3.1: Python Conditional Statements (if, elif, else & Ternary Expressions)

Module 2 Hands-On Capstone Project (100 Marks): Automated Resume & Invoice TextSanitizer CLI — Create a Python text-processing utility that extracts normalized emails, phone numbers, skills, and invoice totals from messy raw text strings using slicing, split/join, and safe type castin...

MODULE 3 OF 16 • STAGE 1 • 11 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Control Statements, Branching & Loop Constructs

Build decision trees and iterative algorithms with if-elif-else, match-case, for-range/enumerate, while loops, and break/continue/pass.

Objectives: ✓ Python Conditional Statements (if, elif, else & Ternary Expressions) — Decision branching, chained comparisons (0 <= score <= 100), nested conditions, and concise conditional expressions. • ✓ Python for Loops, range(), enumerate() & zip() Iteration — Iterating over sequences, range(start, stop, step), index-value pairing with enumerate(), parallel iteration with zip(), and for-else. • ✓ Python while Loops, Sentinel Patterns & Convergence Algorithms — Condition-driven iteration, sentinel input loops, digit/number algorithms, retry backoff counters, and avoiding infinite loops. • ✓ Loop Control Statements — break, continue, pass & Loop-else — Early loop exit with break, skipping invalid iterations with continue, syntactic placeholders with pass, and prime search via for-else. • ✓ Build, verify, and ship Module Capstone: Interactive Banking ATM & Student Grade Analytics Engine.

3.1 Python Conditional Statements (if, elif, else & Ternary Expressions) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: If-elif-else & Ternary • Core Syntax, CLI & Production APIs: if / elif / else, x if cond else y, match / case • Deep-Dive Mechanics: Chained comparison & Truthy check • Advanced Patterns: Nested if & Match-case • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Scholarship & Grade Classifier with Chained Comparisons • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Conditional Statements (if, elif, else & Tern...

Purpose: To understand and apply Python Conditional Statements (if, elif, else & Ternary Expressions) in real Python systems — specifically: Decision branching, chained comparison...

What Topic Is Next → Next Topic 3.2: Python for Loops, range(), enumerate() & zip() Iteration

3.2 Python for Loops, range(), enumerate() & zip() Iteration [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Range() & Enumerate() • Core Syntax, CLI & Production APIs: for item in iterable:, range(), enumerate(), zip() • Deep-Dive Mechanics: Zip() & Iterable • Advanced Patterns: For-else & Nested loop • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Parallel Leaderboard Iteration with enumerate() & zip() • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python for Loops

Purpose: To understand and apply Python for Loops, range(), enumerate() & zip() Iteration in real Python systems — specifically: Iterating over sequences, range(start, stop, step)...

What Topic Is Next → Next Topic 3.3: Python while Loops, Sentinel Patterns & Convergence Algorithms

3.3 Python while Loops, Sentinel Patterns & Convergence Algorithms [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: While condition & Infinite loop • Core Syntax, CLI & Production APIs: while condition:, while-else, counter update • Deep-Dive Mechanics: Sentinel value & While-else • Advanced Patterns: Digit reversal & Retry loop • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Number Reversal & GCD Euclidean Algorithm with while Loop • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python while Loops

Purpose: To understand and apply Python while Loops, Sentinel Patterns & Convergence Algorithms in real Python systems — specifically: Condition-driven iteration, sentinel input l...

What Topic Is Next → Next Topic 3.4: Loop Control Statements — break, continue, pass & Loop-else

3.4 Loop Control Statements — break, continue, pass & Loop-else [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Break & Continue • Core Syntax, CLI & Production APIs: break, continue, pass, for ... else: • Deep-Dive Mechanics: Pass & For-else • Advanced Patterns: Early exit & Prime check • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Prime Finder (for-else) & Noisy Data Filter (continue) •

Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Loop Control Statements

Purpose: To understand and apply Loop Control Statements — break, continue, pass & Loop-else in real Python systems — specifically: Early loop exit with break, skipping invalid it...

What Topic Is Next → Module 3 Capstone (Interactive Banking ATM & Student Grade Analytics Engine) → Next Topic 4.1: Python Lists — Dynamic Arrays, Slicing, Sorting & Shallow vs Deep Copy

Module 3 Hands-On Capstone Project (100 Marks): Interactive Banking ATM & Student Grade Analytics Engine — Develop a menu-driven ATM PIN & transaction simulator alongside a batch student grade classifier using while loops, retry limits, break/continue guards, and summary statistics.

MODULE 4 OF 16 • STAGE 1 • 14 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Core Python Data Structures (Lists, Tuples, Sets & Dictionaries)

Master dynamic arrays (list), immutable records (tuple), hash-based unique sets (set), and O(1) key-value hash tables (dict).

Objectives: ✓ Python Lists — Dynamic Arrays, Slicing, Sorting & Shallow vs Deep Copy — Dynamic resizing arrays, append/extend/insert/pop/remove, custom sort(key=...), and aliasing vs copy.deepcopy(). • ✓ Python Tuples — Immutability, Packing/Unpacking & Hashability — Fixed-size immutable records, single-element trailing comma (x,), returning multiple values, and tuples as dictionary keys. • ✓ Python Sets & Frozensets — Hash Uniqueness & Set Algebra — O(1) membership lookup, automatic deduplication, union (|), intersection (&), difference (-), symmetric_difference (^), and frozenset. • ✓ Python Dictionaries — Hash Maps, Safe .get(), Views & Merging — Insertion-ordered key-value hash maps (Python 3.7+), .get(), .setdefault(), .items(), dict comprehensions, and | merge operator. • ✓ Build, verify, and ship Module Capstone: E-Commerce Inventory, Cart & Deduplicated Coupon Engine.

4.1 Python Lists — Dynamic Arrays, Slicing, Sorting & Shallow vs Deep Copy [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Append vs extend & Timsort • Core Syntax, CLI & Production APIs: .append(), .extend(), .pop(), .sort(key=...), list.copy() • Deep-Dive Mechanics: Shallow copy & Slicing • Advanced Patterns: Mutable sequence & Pop() • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: List Operations, Custom Key Sorting & Matrix Safe Allocation • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Lists

Purpose: To understand and apply Python Lists — Dynamic Arrays, Slicing, Sorting & Shallow vs Deep Copy in real Python systems — specifically: Dynamic resizing arrays, append/exte...

What Topic Is Next → Next Topic 4.2: Python Tuples — Immutability, Packing/Unpacking & Hashability

4.2 Python Tuples — Immutability, Packing/Unpacking & Hashability [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Immutable & Trailing comma • Core Syntax, CLI & Production APIs: (a, b), single = (42,), .count(), .index(), collections.namedtuple • Deep-Dive Mechanics: Hashable key & Unpacking • Advanced Patterns: Namedtuple & Memory efficiency • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Coordinate Grid Lookup Using Hashable Tuple Keys • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Tuples

Purpose: To understand and apply Python Tuples — Immutability, Packing/Unpacking & Hashability in real Python systems — specifically: Fixed-size immutable records, single-element...

What Topic Is Next → Next Topic 4.3: Python Sets & Frozensets — Hash Uniqueness & Set Algebra

4.3 Python Sets & Frozensets — Hash Uniqueness & Set Algebra [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Deduplication & Union • Core Syntax, CLI & Production APIs: set(), .add(), .discard(), |, &, -, ^, frozenset() • Deep-Dive Mechanics: Intersection & Difference • Advanced Patterns: Discard vs remove & Frozenset • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Skill Gap & Deduplication Analyzer Using Set Algebra • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Sets

Purpose: To understand and apply Python Sets & Frozensets — Hash Uniqueness & Set Algebra in real Python systems — specifically: O(1) membership lookup, automatic deduplication, u...

What Topic Is Next → Next Topic 4.4: Python Dictionaries — Hash Maps, Safe .get(), Views & Merging

4.4 Python Dictionaries — Hash Maps, Safe .get(), Views & Merging [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Hash table & Get() • Core Syntax, CLI & Production APIs: d.get(k, default), d.items(), d.setdefault(), d1 | d2, Counter/defaultdict • Deep-Dive Mechanics: Items() & Insertion order • Advanced Patterns: Dict merge | & KeyError • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Word Frequency Counter & Dictionary Merge Pipeline • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Dictionaries

Purpose: To understand and apply Python Dictionaries — Hash Maps, Safe .get(), Views & Merging in real Python systems — specifically: Insertion-ordered key-value hash maps (Python...

What Topic Is Next → Module 4 Capstone (E-Commerce Inventory, Cart & Deduplicated Coupon Engine) → Next Topic 5.1: Python Functions — Parameters, *args, **kwargs, Type Hints & Docstrings

Module 4 Hands-On Capstone Project (100 Marks): E-Commerce Inventory, Cart & Deduplicated Coupon Engine — Build an in-memory store backend using dicts for SKU catalog lookup, lists for ordered cart line-items, tuples for immutable price snapshots, and sets for unique applied coupon codes.

MODULE 5 OF 16 • STAGE 2 • 12 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Modular Python — Functions, Lambdas, Comprehensions & Packages

Write reusable, expressive Python with *args/**kwargs, closures, lambda functional pipelines, comprehensions, and custom packages.

Objectives: ✓ Python Functions — Parameters, *args, **kwargs, Type Hints & Docstrings — Defining functions with def, positional vs keyword arguments, variadic *args/**kwargs, return values, and avoiding mutable default traps. • ✓ Python Lambda Anonymous Functions with map(), filter() & sorted() — Single-expression anonymous functions, custom sort keys, functional transformations with map()/filter()/reduce(), and closure capture. • ✓ List, Dictionary & Set Comprehensions in Python — Declarative sequence construction, filtering clauses, nested matrix flattening, dict inversion, and generator expressions. • ✓ Python Modules, Packages, Standard Library & __name__ Guard — Importing modules, from ... import, sys.path resolution, __init__.py packages, math/random/datetime/collections, and __name__ == "__main__". • ✓ Build, verify, and ship Module Capstone: Modular Employee Payroll, Tax Slab & Analytics Package.

5.1 Python Functions — Parameters, *args, **kwargs, Type Hints & Docstrings [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: *args & **kwargs • Core Syntax, CLI & Production APIs: def fn(a, b=None, *args, **kwargs) -> T: • Deep-Dive Mechanics: Mutable default trap & Keyword arguments • Advanced Patterns: Return tuple & Type hints • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe Default Parameters & Flexible *args / **kwargs Billing Function • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Functions

Purpose: To understand and apply Python Functions — Parameters, *args, **kwargs, Type Hints & Docstrings in real Python systems — specifically: Defining functions with def, positi...

What Topic Is Next → Next Topic 5.2: Python Lambda Anonymous Functions with map(), filter() & sorted()

5.2 Python Lambda Anonymous Functions with map(), filter() & sorted() [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Lambda & Sorted key • Core Syntax, CLI & Production APIs: lambda params: expression, sorted(seq, key=lambda ...), map(), filter() • Deep-Dive Mechanics: Map() & Filter() • Advanced Patterns: Functools.reduce & Late binding • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Multi-Criteria Candidate Ranking with Lambda & filter() • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Lambda Anonymous Functions with map()

Purpose: To understand and apply Python Lambda Anonymous Functions with map(), filter() & sorted() in real Python systems — specifically: Single-expression anonymous functions, cu...

What Topic Is Next → Next Topic 5.3: List, Dictionary & Set Comprehensions in Python

5.3 List, Dictionary & Set Comprehensions in Python [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: List comprehension & Dict comprehension • Core Syntax, CLI & Production APIs: [expr for x in seq if cond], {k: v for ...}, {expr for ...} • Deep-Dive Mechanics: Set comprehension & Flattening • Advanced Patterns: Conditional clause & LIST_APPEND • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Matrix Flattening & Dictionary Inversion via Comprehensions • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for List

Purpose: To understand and apply List, Dictionary & Set Comprehensions in Python in real Python systems — specifically: Declarative sequence construction, filtering clauses, neste...

What Topic Is Next → Next Topic 5.4: Python Modules, Packages, Standard Library & __name__ Guard

5.4 Python Modules, Packages, Standard Library & __name__ Guard [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: __name__ == "__main__" & Sys.modules • Core Syntax, CLI & Production APIs: import module as alias, from pkg import fn, if __name__ == "__main__": • Deep-Dive Mechanics: Standard library & Package • Advanced Patterns: Import alias & Shadowing • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Standard Library math, collections.Counter & __name__ Demo • 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: To understand and apply Python Modules, Packages, Standard Library & __name__ Guard in real Python systems — specifically: Importing modules, from ... import, sys.path re...

What Topic Is Next → Module 5 Capstone (Modular Employee Payroll, Tax Slab & Analytics Package) → Next Topic 6.1: Python Classes, Objects, self, Instance vs Class Attributes

Module 5 Hands-On Capstone Project (100 Marks): Modular Employee Payroll, Tax Slab & Analytics Package — Architect a reusable multi-module Python package that computes net salaries, HRA/PF deductions, progressive tax slabs, and department-wise leaderboard reports using comprehensions and higher...

MODULE 6 OF 16 • STAGE 2 • 14 LIVE + LAB HOURS • 4 TOPICS (32 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Object-Oriented Programming (OOPs) in Python

Design extensible domain models using classes, __init__, instance vs class attributes, super() MRO inheritance, properties, and ABCs.

Objectives: ✓ Python Classes, Objects, self, Instance vs Class Attributes — Blueprints vs instances, explicit self binding, instance __dict__, class attributes, @classmethod, and @staticmethod. • ✓ Python Constructors (__init__ vs __new__), Dunder Methods & Lifecycle — Object allocation (__new__) vs initialization (__init__), parameterized/default constructors, dataclasses, and operator overloading. • ✓ Inheritance in Python — super(), Method Overriding & MRO — Single, multilevel, hierarchical, and multiple inheritance, cooperative super(), Method Resolution Order (__mro__), and polymorphism. • ✓ Encapsulation, @property Accessors & Abstract Base Classes (abc) — Protected (_attr) vs name-mangled private (__attr) members, Pythonic @property/@setter validation, and abc.ABC contracts. • ✓ Build, verify, and ship Module Capstone: OOP Digital Library & Multi-Tier Banking Ledger System.

6.1 Python Classes, Objects, self, Instance vs Class Attributes [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Self & Class attribute • Core Syntax, CLI & Production APIs: class Name:, self, @classmethod, @staticmethod, __str__ / __repr__ • Deep-Dive Mechanics: Instance attribute & @classmethod • Advanced Patterns: @staticmethod & __repr__ • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Student Enrollment Class with @classmethod Factory • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Classes

Purpose: To understand and apply Python Classes, Objects, self, Instance vs Class Attributes in real Python systems — specifically: Blueprints vs instances, explicit self binding,...

What Topic Is Next → Next Topic 6.2: Python Constructors (__init__ vs __new__), Dunder Methods & Lifecycle

6.2 Python Constructors (__init__ vs __new__), Dunder Methods & Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: __init__ & __new__ • Core Syntax, CLI & Production APIs: def __init__(self, ...):, __new__, __eq__, __len__, @dataclass • Deep-Dive Mechanics: Dunder methods & Dataclass • Advanced Patterns: Object lifecycle & __eq__ • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Custom Vector Class with __init__, __add__ & __eq__ • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Constructors (__init__ vs __new__)

Purpose: To understand and apply Python Constructors (__init__ vs __new__), Dunder Methods & Lifecycle in real Python systems — specifically: Object allocation (__new__) vs initia...

What Topic Is Next → Next Topic 6.3: Inheritance in Python — super(), Method Overriding & MRO

6.3 Inheritance in Python — super(), Method Overriding & MRO [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Super() & Method overriding • Core Syntax, CLI & Production APIs: class Child(Parent):, super().__init__(), Class.mro(), issubclass() • Deep-Dive Mechanics: MRO & Polymorphism • Advanced Patterns: Multiple inheritance & Issubclass • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Polymorphic Payment Gateway Inheritance with super() • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Inheritance in Python

Purpose: To understand and apply Inheritance in Python — super(), Method Overriding & MRO in real Python systems — specifically: Single, multilevel, hierarchical, and multiple inh...

What Topic Is Next → Next Topic 6.4: Encapsulation, @property Accessors & Abstract Base Classes (abc)

6.4 Encapsulation, @property Accessors & Abstract Base Classes (abc) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: @property & Name mangling • Core Syntax, CLI & Production APIs: @property, @attr.setter, from abc import ABC, abstractmethod • Deep-Dive Mechanics: ABC & @abstractmethod • Advanced Patterns: Encapsulation & Invariant • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Encapsulated Wallet with @property Validation & ABC • 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: To understand and apply Encapsulation, @property Accessors & Abstract Base Classes (abc) in real Python systems — specifically: Protected (_attr) vs name-mangled private...

What Topic Is Next → Module 6 Capstone (OOP Digital Library & Multi-Tier Banking Ledger System) → Next Topic 7.1: Python Exception Handling — try, except, else, finally & Custom Errors

Module 6 Hands-On Capstone Project (100 Marks): OOP Digital Library & Multi-Tier Banking Ledger System — Design an object-oriented Banking & Library domain featuring abstract base accounts, Savings/Current subclasses with overdraft rules, @property balance guards, and transaction audit logs.

MODULE 7 OF 16 • STAGE 2 • 11 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Exception Handling, File I/O, CSV & JSON Serialization

Build resilient programs with try-except-else-finally, custom exceptions, context managers (with open), pathlib, CSV, and JSON.

Objectives: ✓ Python Exception Handling — try, except, else, finally & Custom Errors — Exception hierarchy, catching specific errors, else/finally execution guarantees, raise ... from ..., and custom Exception classes. • ✓ Python File Handling — Context Managers (with open), Modes & pathlib — File modes (r, w, a, rb), automatic descriptor cleanup via with open(..., encoding="utf-8"), line streaming, csv module, and pathlib.Path. • ✓ Working with JSON in Python — loads(), dumps(), Custom Encoders & APIs — Mapping Python dicts/lists to JSON objects/arrays, json.loads() vs json.load(), json.dumps(indent=2, sort_keys=True), and handling JSONDecodeError. • ✓ Build, verify, and ship Module Capstone: Fault-Tolerant Server Log Analyzer & JSON/CSV ETL Pipeline.

7.1 Python Exception Handling — try, except, else, finally & Custom Errors [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Try-except & Finally • Core Syntax, CLI & Production APIs: try / except ValueError as e / else / finally / raise • Deep-Dive Mechanics: Else block & Raise • Advanced Patterns: Custom Exception & Exception hierarchy • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Custom Domain Exception & try-except-else-finally Flow • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Exception Handling

Purpose: To understand and apply Python Exception Handling — try, except, else, finally & Custom Errors in real Python systems — specifically: Exception hierarchy, catching specif...

What Topic Is Next → Next Topic 7.2: Python File Handling — Context Managers (with open), Modes & pathlib

7.2 Python File Handling — Context Managers (with open), Modes & pathlib [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: With open & File modes • Core Syntax, CLI & Production APIs: with open(path, mode, encoding="utf-8") as f:, pathlib.Path • Deep-Dive Mechanics: UTF-8 encoding & Line streaming • Advanced Patterns: Pathlib & Csv • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: In-Memory Stream & Safe Context Manager Line Processor • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python File Handling

Purpose: To understand and apply Python File Handling — Context Managers (with open), Modes & pathlib in real Python systems — specifically: File modes (r, w, a, rb), automatic de...

What Topic Is Next → Next Topic 7.3: Working with JSON in Python — loads(), dumps(), Custom Encoders & APIs

7.3 Working with JSON in Python — loads(), dumps(), Custom Encoders & APIs [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Json.loads & Json.dumps • Core Syntax, CLI & Production APIs: json.loads(), json.dumps(), json.load(), json.dump(), JSONDecodeError • Deep-Dive Mechanics: JSONDecodeError & Load vs loads • Advanced Patterns: Serialization & Type mapping • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: JSON API Payload Parsing & Deterministic Serialization • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Working with JSON in Python

Purpose: To understand and apply Working with JSON in Python — loads(), dumps(), Custom Encoders & APIs in real Python systems — specifically: Mapping Python dicts/lists to JSON o...

What Topic Is Next → Module 7 Capstone (Fault-Tolerant Server Log Analyzer & JSON/CSV ETL Pipeline) → Next Topic 8.1: Linear & Binary Search in Python (Iterative, Recursive & bisect)

Module 7 Hands-On Capstone Project (100 Marks): Fault-Tolerant Server Log Analyzer & JSON/CSV ETL Pipeline — Create a Python ETL pipeline that streams raw application log files, isolates malformed lines into an error audit log without crashing, aggregates HTTP status counts, and exports clean JSON...

MODULE 8 OF 16 • STAGE 2 • 10 LIVE + LAB HOURS • 2 TOPICS (16 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Searching & Sorting Algorithms in Python (DSA)

Crack coding rounds with iterative/recursive Binary Search, bisect, Merge Sort, Quick Sort, and Python Timsort complexity analysis.

Objectives: ✓ Linear & Binary Search in Python (Iterative, Recursive & bisect) — Halving sorted search spaces in O(log n), leftmost/rightmost insertion bounds, and standard library bisect_left / bisect_right. • ✓ Divide & Conquer Sorting — Merge Sort, Quick Sort & Timsort Internals — Stable O(n log n) Merge Sort, partition-based Quick Sort pivot selection, stability guarantees, and how CPython Timsort exploits natural runs. • ✓ Build, verify, and ship Module Capstone: High-Speed Product Catalog Search & Sorting Benchmark Suite.

8.1 Linear & Binary Search in Python (Iterative, Recursive & bisect) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: O(log n) & Sorted precondition • Core Syntax, CLI & Production APIs: low <= high, mid = (low + high) // 2, bisect.bisect_left() • Deep-Dive Mechanics: Mid calculation & Bisect_left • Advanced Patterns: Off-by-one & Search space • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Iterative Binary Search & bisect Boundary Finder • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Linear

Purpose: To understand and apply Linear & Binary Search in Python (Iterative, Recursive & bisect) in real Python systems — specifically: Halving sorted search spaces in O(log n),...

What Topic Is Next → Next Topic 8.2: Divide & Conquer Sorting — Merge Sort, Quick Sort & Timsort Internals

8.2 Divide & Conquer Sorting — Merge Sort, Quick Sort & Timsort Internals [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Divide and conquer & O(n log n) • Core Syntax, CLI & Production APIs: merge_sort(), partition pivot, sorted() / list.sort() Timsort • Deep-Dive Mechanics: Sort stability & Pivot partition • Advanced Patterns: Timsort & Auxiliary space • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Stable Merge Sort Implementation in Python 3 • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Divide

Purpose: To understand and apply Divide & Conquer Sorting — Merge Sort, Quick Sort & Timsort Internals in real Python systems — specifically: Stable O(n log n) Merge Sort, partiti...

What Topic Is Next → Module 8 Capstone (High-Speed Product Catalog Search & Sorting Benchmark Suite) → Next Topic 9.1: Python Closures & @Decorators (functools.wraps, Timing & Auth Guards)

Module 8 Hands-On Capstone Project (100 Marks): High-Speed Product Catalog Search & Sorting Benchmark Suite — Build an algorithmic search and ranking benchmark tool on 50,000 synthetic e-commerce products comparing Linear Scan vs Binary Search (bisect) and Merge Sort vs Quick Sort vs native Timsort.

MODULE 9 OF 16 • STAGE 3 • 13 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Advanced Python — Decorators, Generators, Regex & Web Scraping

Write senior-level Python with higher-order @decorators, memory-efficient yield generators, compiled regular expressions, and HTML scraping.

Objectives: ✓ Python Closures & @Decorators (functools.wraps, Timing & Auth Guards) — First-class functions, lexical closures, @decorator syntactic sugar, wrapping *args/**kwargs, parameterized decorators, and @functools.wraps. • ✓ Python Iterators & Generators — yield, next(), StopIteration & Lazy Streams — Iterator protocol (__iter__/__next__), state-suspending yield functions, StopIteration, yield from, and O(1) memory streaming. • ✓ Python Regular Expressions (re Module) & Structured HTML Extraction — Raw strings r"...", character classes, quantifiers, capture groups, re.search/findall/sub/compile, and extracting clean data from HTML. • ✓ Build, verify, and ship Module Capstone: Automated Tech Job & Price Tracker Web Scraper with Regex Cleaner.

9.1 Python Closures & @Decorators (functools.wraps, Timing & Auth Guards) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Closure & @decorator • Core Syntax, CLI & Production APIs: @decorator, @functools.wraps(func), wrapper(*args, **kwargs) • Deep-Dive Mechanics: Functools.wraps & Higher-order function • Advanced Patterns: *args **kwargs & Nonlocal • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Reusable Execution Audit & Validation Decorator with wraps • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Closures

Purpose: To understand and apply Python Closures & @Decorators (functools.wraps, Timing & Auth Guards) in real Python systems — specifically: First-class functions, lexical closur...

What Topic Is Next → Next Topic 9.2: Python Iterators & Generators — yield, next(), StopIteration & Lazy Streams

9.2 Python Iterators & Generators — yield, next(), StopIteration & Lazy Streams [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Yield & StopIteration • Core Syntax, CLI & Production APIs: yield, next(gen, default), yield from, (x for x in seq) • Deep-Dive Mechanics: Lazy evaluation & Single-pass exhaustion • Advanced Patterns: Next() & Yield from • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Memory-Efficient Fibonacci & Chunked Batch Generator • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Iterators

Purpose: To understand and apply Python Iterators & Generators — yield, next(), StopIteration & Lazy Streams in real Python systems — specifically: Iterator protocol (__iter__/__n...

What Topic Is Next → Next Topic 9.3: Python Regular Expressions (re Module) & Structured HTML Extraction

9.3 Python Regular Expressions (re Module) & Structured HTML Extraction [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Raw string r"" & Re.findall • Core Syntax, CLI & Production APIs: re.compile(r"..."), re.search(), re.findall(), re.sub(), html.parser • Deep-Dive Mechanics: Re.sub & Capture groups • Advanced Patterns: Greedy vs non-greedy & Re.compile • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Regex Email, Price & HTML Tag Extractor Pipeline • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Regular Expressions (re Module)

Purpose: To understand and apply Python Regular Expressions (re Module) & Structured HTML Extraction in real Python systems — specifically: Raw strings r"...", character classes,...

What Topic Is Next → Module 9 Capstone (Automated Tech Job & Price Tracker Web Scraper with Regex Cleaner) → Next Topic 10.1: Python MySQL Database Connectivity, Parameterized Queries & Transactions

Module 9 Hands-On Capstone Project (100 Marks): Automated Tech Job & Price Tracker Web Scraper with Regex Cleaner — Build an ethical Python scraper and HTML parser that extracts course/job titles, cleans messy salary/price strings via compiled regular expressions, streams records lazily via generators, an...

MODULE 10 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Python Databases (MySQL, SQLite3) & Tkinter Desktop GUI

Connect Python applications to relational SQL databases safely using parameterized DB-API 2.0 queries and build Tkinter desktop interfaces.

Objectives: ✓ Python MySQL Database Connectivity, Parameterized Queries & Transactions — PEP-249 DB-API 2.0 workflow, mysql-connector/PyMySQL connections, %s parameterized placeholders, fetchone/fetchall, and commit/rollback. • ✓ Python SQLite3 — Embedded Relational Database, Row Factory & CRUD — Zero-configuration built-in sqlite3 module, :memory: vs file DBs, ? parameter placeholders, sqlite3.Row dict-style access, and ACID transactions. • ✓ Python Tkinter GUI — Widgets, Grid Geometry, StringVar & Event Callbacks — Building cross-platform desktop apps with Tk(), Label/Entry/Button/ttk.Treeview, grid() vs pack(), StringVar two-way binding, and non-blocking callbacks. • ✓ Build, verify, and ship Module Capstone: Desktop Student Admission, Fee Receipt & Attendance Manager (Tkinter + SQLite3).

10.1 Python MySQL Database Connectivity, Parameterized Queries & Transactions [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: DB-API 2.0 & SQL injection prevention • Core Syntax, CLI & Production APIs: conn.cursor(), cursor.execute(sql, params), conn.commit(), conn.rollback() • Deep-Dive Mechanics: Parameterized %s & Commit and rollback • Advanced Patterns: Cursor.fetchall & Connection pooling • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Safe Parameterized SQL Query Builder & Transaction Pattern • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python MySQL Database Connectivity

Purpose: To understand and apply Python MySQL Database Connectivity, Parameterized Queries & Transactions in real Python systems — specifically: PEP-249 DB-API 2.0 workflow, mysql...

What Topic Is Next → Next Topic 10.2: Python SQLite3 — Embedded Relational Database, Row Factory & CRUD

10.2 Python SQLite3 — Embedded Relational Database, Row Factory & CRUD [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Sqlite3.connect & ? placeholder • Core Syntax, CLI & Production APIs: sqlite3.connect(), cursor.execute("... WHERE id = ?", (id,)), sqlite3.Row • Deep-Dive Mechanics: Sqlite3.Row & :memory: • Advanced Patterns: Executemany & ACID transaction • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Complete Runnable In-Memory SQLite3 CRUD Program • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python SQLite3

Purpose: To understand and apply Python SQLite3 — Embedded Relational Database, Row Factory & CRUD in real Python systems — specifically: Zero-configuration built-in sqlite3 modul...

What Topic Is Next → Next Topic 10.3: Python Tkinter GUI — Widgets, Grid Geometry, StringVar & Event Callbacks

10.3 Python Tkinter GUI — Widgets, Grid Geometry, StringVar & Event Callbacks [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Subtopics Covered: Mainloop() & Grid vs pack • Core Syntax, CLI & Production APIs: tk.Tk(), ttk.Frame, .grid(row, column), tk.StringVar(), root.mainloop() • Deep-Dive Mechanics: StringVar & Command callback • Advanced Patterns: Ttk.Treeview & Root.after() • Production Failure Mode & Pitfall Guard • Hands-On Live Coding Lab: Tkinter Fee Calculator Controller Pattern (Testable View-Model) • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Tkinter GUI

Purpose: To understand and apply Python Tkinter GUI — Widgets, Grid Geometry, StringVar & Event Callbacks in real Python systems — specifically: Building cross-platform desktop ap...

What Topic Is Next → Module 10 Capstone (Desktop Student Admission, Fee Receipt & Attendance Manager (Tkinter + SQLite3)) → Next Topic 11.1: async/await Coroutines, Tasks & asyncio.gather()

Module 10 Hands-On Capstone Project (100 Marks): Desktop Student Admission, Fee Receipt & Attendance Manager (Tkinter + SQLite3) — Build a complete standalone desktop application using Tkinter (ttk Treeview + forms) backed by a persistent SQLite3/MySQL relational database with parameterized CRUD queries, search filters,...

MODULE 11 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Python Asyncio Event Loop, Coroutines & Concurrent Network I/O

Production-grade Python Training engineering module covering Python Asyncio Event Loop, Coroutines & Concurrent Network I/O with hands-on labs and capstone build.

Objectives: ✓ async/await Coroutines, Tasks & asyncio.gather() — Fetch 100+ partner HTTP endpoints concurrently in 200ms instead of 20 sequential seconds. • ✓ Async Context Managers (async with) & Semaphores — Bound concurrent outbound sockets cleanly so Python scripts never trigger 429 rate-limit bans. • ✓ Threading vs Multiprocessing vs Python GIL Offloading — Use ProcessPoolExecutor for CPU-heavy parsing and asyncio for high-volume network I/O. • ✓ Build, verify, and ship Module Capstone: Async Multi-Endpoint API Scraper & Webhook Dispatcher.

11.1 async/await Coroutines, Tasks & asyncio.gather() [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Tasks • Integration with Async Context Managers (async with) & Semaphores • Live Debugging, Telemetry & Technical Interview Trade • What Is async/await Coroutines, Tasks & asyncio.gather() & Why Industry Teams Use It • In Python Training, async/await Coroutines, Tasks & asyncio.gather() exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply async/await Coroutines, Tasks & asyncio.gather() in real Python systems — specifically: Fetch 100+ partner HTTP endpoints concurrently in 200ms in...

What Topic Is Next → Next Topic 11.2: Async Context Managers (async with) & Semaphores

11.2 Async Context Managers (async with) & Semaphores [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Semaphores • Integration with Threading vs Multiprocessing vs Python GIL Offloading • Live Debugging, Telemetry & Technical Interview Trade • What Is Async Context Managers (async with) & Semaphores & Why Industry Teams Use It • In Python Training, Async Context Managers (async with) & Semaphores exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Async Context Managers (async with) & Semaphores in real Python systems — specifically: Bound concurrent outbound sockets cleanly so Python script...

What Topic Is Next → Next Topic 11.3: Threading vs Multiprocessing vs Python GIL Offloading

11.3 Threading vs Multiprocessing vs Python GIL Offloading [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with async/await Coroutines, Tasks & asyncio.gather() • Live Debugging, Telemetry & Technical Interview Trade • What Is Threading vs Multiprocessing vs Python GIL Offloading & Why Industry Teams Use It • In Python Training, Threading vs Multiprocessing vs Python GIL Offloading exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Threading vs Multiprocessing vs Python GIL Offloading in real Python systems — specifically: Use ProcessPoolExecutor for CPU-heavy parsing and asy...

What Topic Is Next → Module 11 Capstone (Async Multi-Endpoint API Scraper & Webhook Dispatcher) → Next Topic 12.1: Python Type Hints, Generics, Protocol & mypy Verification

Module 11 Hands-On Capstone Project (100 Marks): Async Multi-Endpoint API Scraper & Webhook Dispatcher — Design, implement, test, and document 'Async Multi-Endpoint API Scraper & Webhook Dispatcher' applying every production concept from Module 11 (Python Asyncio Event Loop, Coroutines & Concur...

MODULE 12 OF 16 • STAGE 3 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Static Type Checking (mypy), Dataclasses, Pydantic & Clean Packaging

Production-grade Python Training engineering module covering Static Type Checking (mypy), Dataclasses, Pydantic & Clean Packaging with hands-on labs and capstone build.

Objectives: ✓ Python Type Hints, Generics, Protocol & mypy Verification — Catch NoneType and dict key typos before runtime across large multi-developer Python repos. • ✓ Immutable @dataclass(frozen=True, slots=True) & Enum — Cut per-object RAM usage by 40% and prevent accidental state mutation across functions. • ✓ Modern Packaging with pyproject.toml, Virtualenvs & CLI Entrypoints — Publish clean, reproducible Python CLI tools and internal wheels for enterprise teams. • ✓ Build, verify, and ship Module Capstone: Type-Safe Python Domain SDK & Installable pyproject.toml Package.

12.1 Python Type Hints, Generics, Protocol & mypy Verification [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Generics • Integration with Immutable @dataclass(frozen=True, slots=True) & Enum • Live Debugging, Telemetry & Technical Interview Trade • What Is Python Type Hints, Generics, Protocol & mypy Verification & Why Industry Teams Use It • In Python Training, Python Type Hints, Generics, Protocol & mypy Verification exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Python Type Hints, Generics, Protocol & mypy Verification in real Python systems — specifically: Catch NoneType and dict key typos before runtime...

What Topic Is Next → Next Topic 12.2: Immutable @dataclass(frozen=True, slots=True) & Enum

12.2 Immutable @dataclass(frozen=True, slots=True) & Enum [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of slots=True) • Integration with Modern Packaging with pyproject.toml, Virtualenvs & CLI Entrypoints • Live Debugging, Telemetry & Technical Interview Trade • What Is Immutable @dataclass(frozen=True, slots=True) & Enum & Why Industry Teams Use It • In Python Training, Immutable @dataclass(frozen=True, slots=True) & Enum exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Immutable @dataclass(frozen=True, slots=True) & Enum in real Python systems — specifically: Cut per-object RAM usage by 40% and prevent accidental...

What Topic Is Next → Next Topic 12.3: Modern Packaging with pyproject.toml, Virtualenvs & CLI Entrypoints

12.3 Modern Packaging with pyproject.toml, Virtualenvs & CLI Entrypoints [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Virtualenvs • Integration with Python Type Hints, Generics, Protocol & mypy Verification • Live Debugging, Telemetry & Technical Interview Trade • What Is Modern Packaging with pyproject.toml, Virtualenvs & CLI Entrypoints & Why Industry Teams Use It • In Python Training, Modern Packaging with pyproject.toml, Virtualenvs & CLI Entrypoints exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Modern Packaging with pyproject.toml, Virtualenvs & CLI Entrypoints in real Python systems — specifically: Publish clean, reproducible Python CLI...

What Topic Is Next → Module 12 Capstone (Type-Safe Python Domain SDK & Installable pyproject.toml Package) → Next Topic 13.1: Pytest Fixtures, Scopes (function/module/session) & tmp_path

Module 12 Hands-On Capstone Project (100 Marks): Type-Safe Python Domain SDK & Installable pyproject.toml Package — Design, implement, test, and document 'Type-Safe Python Domain SDK & Installable pyproject.toml Package' applying every production concept from Module 12 (Static Type Checking (mypy), Datacl...

MODULE 13 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Automated Testing with Pytest: Fixtures, Parametrize & Mocking

Production-grade Python Training engineering module covering Automated Testing with Pytest: Fixtures, Parametrize & Mocking with hands-on labs and capstone build.

Objectives: ✓ Pytest Fixtures, Scopes (function/module/session) & tmp_path — Share clean database and temp-file setups across hundreds of tests without boilerplate. • ✓ @pytest.mark.parametrize Edge-Case Matrices — Test 20 boundary inputs (empty strings, negative ₹, Unicode, None) in a single 8-line test. • ✓ monkeypatch & unittest.mock for External API/Clock Isolation — Test retry logic and billing cutoffs deterministically without calling live servers. • ✓ Build, verify, and ship Module Capstone: 95%-Coverage Pytest Suite with Deterministic Gateway Mocks.

13.1 Pytest Fixtures, Scopes (function/module/session) & tmp_path [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Scopes (function/module/session) • Integration with @pytest.mark.parametrize Edge • Live Debugging, Telemetry & Technical Interview Trade • What Is Pytest Fixtures, Scopes (function/module/session) & tmp_path & Why Industry Teams Use It • In Python Training, Pytest Fixtures, Scopes (function/module/session) & tmp_path exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Pytest Fixtures, Scopes (function/module/session) & tmp_path in real Python systems — specifically: Share clean database and temp-file setups acro...

What Topic Is Next → Next Topic 13.2: @pytest.mark.parametrize Edge-Case Matrices

13.2 @pytest.mark.parametrize Edge-Case Matrices [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Edge • Integration with monkeypatch & unittest.mock for External API/Clock Isolation • Live Debugging, Telemetry & Technical Interview Trade • What Is @pytest.mark.parametrize Edge • In Python Training, @pytest.mark.parametrize Edge • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply @pytest.mark.parametrize Edge-Case Matrices in real Python systems — specifically: Test 20 boundary inputs (empty strings, negative ₹, Unicode, No...

What Topic Is Next → Next Topic 13.3: monkeypatch & unittest.mock for External API/Clock Isolation

13.3 monkeypatch & unittest.mock for External API/Clock Isolation [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of unittest.mock for External API/Clock Isolation • Integration with Pytest Fixtures, Scopes (function/module/session) & tmp_path • Live Debugging, Telemetry & Technical Interview Trade • What Is monkeypatch & unittest.mock for External API/Clock Isolation & Why Industry Teams Use It • In Python Training, monkeypatch & unittest.mock for External API/Clock Isolation exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply monkeypatch & unittest.mock for External API/Clock Isolation in real Python systems — specifically: Test retry logic and billing cutoffs determini...

What Topic Is Next → Module 13 Capstone (95%-Coverage Pytest Suite with Deterministic Gateway Mocks) → Next Topic 14.1: Lazy Generators (yield / yield from) & itertools Pipelines

Module 13 Hands-On Capstone Project (100 Marks): 95%-Coverage Pytest Suite with Deterministic Gateway Mocks — Design, implement, test, and document '95%-Coverage Pytest Suite with Deterministic Gateway Mocks' applying every production concept from Module 13 (Automated Testing with Pytest: Fixtures,...

MODULE 14 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Python Memory Profiling, Generators, Security & Production Logging

Production-grade Python Training engineering module covering Python Memory Profiling, Generators, Security & Production Logging with hands-on labs and capstone build.

Objectives: ✓ Lazy Generators (yield / yield from) & itertools Pipelines — Process 10GB CSV/log files line-by-line using < 25MB constant Python RAM. • ✓ Secure Secrets Handling, Subprocess Safety & SQL Parameterization — Eliminate command-injection, pickle-deserialization, and hardcoded-key vulnerabilities. • ✓ Structured JSON Logging, Rotating Handlers & Custom Exception Hierarchies — Emit searchable production logs with correlation IDs and clean error recovery. • ✓ Build, verify, and ship Module Capstone: Zero-Leak Streaming Log Processor & Audit CLI.

14.1 Lazy Generators (yield / yield from) & itertools Pipelines [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of itertools Pipelines • Integration with Secure Secrets Handling, Subprocess Safety & SQL Parameterization • Live Debugging, Telemetry & Technical Interview Trade • What Is Lazy Generators (yield / yield from) & itertools Pipelines & Why Industry Teams Use It • In Python Training, Lazy Generators (yield / yield from) & itertools Pipelines exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Lazy Generators (yield / yield from) & itertools Pipelines in real Python systems — specifically: Process 10GB CSV/log files line-by-line using <...

What Topic Is Next → Next Topic 14.2: Secure Secrets Handling, Subprocess Safety & SQL Parameterization

14.2 Secure Secrets Handling, Subprocess Safety & SQL Parameterization [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Subprocess Safety • Integration with Structured JSON Logging, Rotating Handlers & Custom Exception Hierarchies • Live Debugging, Telemetry & Technical Interview Trade • What Is Secure Secrets Handling, Subprocess Safety & SQL Parameterization & Why Industry Teams Use It • In Python Training, Secure Secrets Handling, Subprocess Safety & SQL Parameterization exists for one clear real • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Secure Secrets Handling, Subprocess Safety & SQL Parameterization in real Python systems — specifically: Eliminate command-injection, pickle-deser...

What Topic Is Next → Next Topic 14.3: Structured JSON Logging, Rotating Handlers & Custom Exception Hierarchies

14.3 Structured JSON Logging, Rotating Handlers & Custom Exception Hierarchies [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Plain • Production Implementation of Rotating Handlers • Integration with Lazy Generators (yield / yield from) & itertools Pipelines • Live Debugging, Telemetry & Technical Interview Trade • What Is Structured JSON Logging, Rotating Handlers & Custom Exception Hierarchies & Why Industry Teams Use It • In Python Training, Structured JSON Logging, Rotating Handlers & Custom Exception Hierarchies exists for one clear.... • Even if you come from a non • Runnable Code Implementation (PYTHON)

Purpose: To understand and apply Structured JSON Logging, Rotating Handlers & Custom Exception Hierarchies in real Python systems — specifically: Emit searchable production logs w...

What Topic Is Next → Module 14 Capstone (Zero-Leak Streaming Log Processor & Audit CLI) → Next Topic 15.1: Strict Type Hints, Protocols, TypedDict & Static Checking with mypy/pyright

Module 14 Hands-On Capstone Project (100 Marks): Zero-Leak Streaming Log Processor & Audit CLI — Design, implement, test, and document 'Zero-Leak Streaming Log Processor & Audit CLI' applying every production concept from Module 14 (Python Memory Profiling, Generators, Security & Produc...

MODULE 15 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Modern Python Type Safety, Packaging & CLI/Automation Engineering

Self-explanatory list-and-bullet production mastery module covering Modern Python Type Safety, Packaging & CLI/Automation Engineering in Python Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Strict Type Hints, Protocols, TypedDict & Static Checking with mypy/pyright — Purpose: Catch subtle NoneType and schema bugs in editors and CI pipelines before Python code ever runs. • ✓ Modern pyproject.toml Packaging, Virtual Environments (uv/venv) & Ruff Linting — Purpose: Structure reproducible, dependency-pinned Python projects that install cleanly on any server. • ✓ Resilient CLI Tools, Structured Logging & Retry/Timeout HTTP Automation — Purpose: Automate enterprise file, API, and reporting workflows with clean exit codes and audit logs. • ✓ Build, verify, and document 'Typed Python CLI Data Validator & Wheel Package with ruff + mypy CI' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

15.1 Strict Type Hints, Protocols, TypedDict & Static Checking with mypy/pyright [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Strict Type Hints, Protocols, TypedDict & Static Checking with mypy/pyrigh... • Performance Optimization, Complexity & Latency Impact of Strict Type Hints, Protocols, TypedDict & Static Checking.... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Strict Type Hints • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: To understand and apply Strict Type Hints, Protocols, TypedDict & Static Checking with mypy/pyright in real Python systems — specifically: Catch subtle NoneType and schem...

What Topic Is Next → Next Topic 15.2: Modern pyproject.toml Packaging, Virtual Environments (uv/venv) & Ruff Linting

15.2 Modern pyproject.toml Packaging, Virtual Environments (uv/venv) & Ruff Linting [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Modern pyproject.toml Packaging, Virtual Environments (uv/venv) & Ruff Lin... • Performance Optimization, Complexity & Latency Impact of Modern pyproject.toml Packaging, Virtual Environments (uv/... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Modern pyproject.toml Packaging • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: To understand and apply Modern pyproject.toml Packaging, Virtual Environments (uv/venv) & Ruff Linting in real Python systems — specifically: Structure reproducible, depe...

What Topic Is Next → Next Topic 15.3: Resilient CLI Tools, Structured Logging & Retry/Timeout HTTP Automation

15.3 Resilient CLI Tools, Structured Logging & Retry/Timeout HTTP Automation [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Resilient CLI Tools, Structured Logging & Retry/Timeout HTTP Automation • Performance Optimization, Complexity & Latency Impact of Resilient CLI Tools, Structured Logging & Retry/Timeout HT... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for

Resilient CLI Tools • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: To understand and apply Resilient CLI Tools, Structured Logging & Retry/Timeout HTTP Automation in real Python systems — specifically: Automate enterprise file, API, and...

What Topic Is Next → Module 15 Capstone (Typed Python CLI Data Validator & Wheel Package with ruff + mypy CI) → Next Topic 16.1: Memory Profiling with tracemalloc, __slots__ & Generator Streaming Pipelines

Module 15 Hands-On Capstone Project (100 Marks): Typed Python CLI Data Validator & Wheel Package with ruff + mypy CI — Architect, implement, test, and document 'Typed Python CLI Data Validator & Wheel Package with ruff + mypy CI' applying every bullet-wise engineering concept from Module 15 (Modern Python Ty...

MODULE 16 OF 16 • STAGE 4 • 14 LIVE + LAB HOURS • 3 TOPICS (24 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Python Memory Profiling, Security Hygiene & Coding Interview Mastery

Self-explanatory list-and-bullet production mastery module covering Python Memory Profiling, Security Hygiene & Coding Interview Mastery in Python Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken MockAttempt AI Viva.

Objectives: ✓ Memory Profiling with tracemalloc, __slots__ & Generator Streaming Pipelines — Purpose: Process multi-gigabyte CSV/JSON datasets in constant RAM without Out-Of-Memory crashes. • ✓ Secure Python Practices: Subprocess Safety, Secrets Handling & Safe Deserialization — Purpose: Prevent command injection, pickle exploits, and credential leaks in production Python scripts. • ✓ Python Internals, Edge-Case Traps & Live Technical Interview Problem Solving — Purpose: Master mutable default traps, GIL trade-offs, and clean Pythonic algorithmic solutions. • ✓ Build, verify, and document 'tracemalloc Memory Auditor & High-Throughput Stream Transformation Engine' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + MockAttempt AI Technical Interviews.

16.1 Memory Profiling with tracemalloc, __slots__ & Generator Streaming Pipelines [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Memory Profiling with tracemalloc, __slots__ & Generator Streaming Pipelin... • Performance Optimization, Complexity & Latency Impact of Memory Profiling with tracemalloc, __slots__ & Generator S... • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Memory Profiling with tracemalloc • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: To understand and apply Memory Profiling with tracemalloc, __slots__ & Generator Streaming Pipelines in real Python systems — specifically: Process multi-gigabyte CSV/JSO...

What Topic Is Next → Next Topic 16.2: Secure Python Practices: Subprocess Safety, Secrets Handling & Safe Deserialization

16.2 Secure Python Practices: Subprocess Safety, Secrets Handling & Safe Deserialization [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Secure Python Practices • Performance Optimization, Complexity & Latency Impact of Secure Python Practices • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Secure Python Practices • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: To understand and apply Secure Python Practices: Subprocess Safety, Secrets Handling & Safe Deserialization in real Python systems — specifically: Prevent command injecti...

What Topic Is Next → Next Topic 16.3: Python Internals, Edge-Case Traps & Live Technical Interview Problem Solving

16.3 Python Internals, Edge-Case Traps & Live Technical Interview Problem Solving [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Subtopics Covered: Security Hygiene & Data Safety Rules for Python Internals, Edge • Performance Optimization, Complexity & Latency Impact of Python Internals, Edge • Live Debugging, Stack • Runnable Code Implementation (PYTHON) • Hands • Idiomatic PYTH Implementation: PEP-8 type hints, generators, and O(1) dict/set structures • CPython 3.12 Bytecode & Heap Execution & Memory Lifecycle for Python Internals • Production Hardening & Security: input sanitization, safe deserialization & exception hierarchy

Purpose: To understand and apply Python Internals, Edge-Case Traps & Live Technical Interview Problem Solving in real Python systems — specifically: Master mutable default traps,...

What Topic Is Next → Module 16 Capstone → End-of-Course Live Client Project: DataFlow Automated ETL Reconciliation, CLI Ops & Reporting Engine

Module 16 Hands-On Capstone Project (100 Marks): tracemalloc Memory Auditor & High-Throughput Stream Transformation Engine — Architect, implement, test, and document 'tracemalloc Memory Auditor & High-Throughput Stream Transformation Engine' applying every bullet-wise engineering concept from Module 16 (Python Mem...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

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

Project: DataFlow Automated ETL Reconciliation, CLI Ops & Reporting Engine • **Confirmed Role:** Python Software & Automation Engineering Intern

Domain & Production Infra: FinTech Accounting & Automated Back-Office Operations Client (Python 3.12 + Poetry/uv + Pydantic + SQLite/PostgreSQL + Scheduled Linux Cron/Docker)

- Engineered modular Python 3.12 packages using strict type annotations, dataclasses, custom exceptions, and context managers.

- Built memory-safe generator pipelines processing 500k+ CSV/JSON ledger records with automated anomaly detection.
- Integrated REST API clients with retry/backoff policies, SQLite/PostgreSQL persistence, and structured logging.
- Shipped pytest unit suites (90%+ coverage) and containerized CLI workflows to live client production.

Target Job Roles: Junior Python Developer, Backend Automation Engineer, Python Scripting & QA Automation Analyst, Associate Software Engineer (Python)

Realistic Annual CTC Scope: **Fresher / Entry-Level (0–1 Yr):** ₹4.5 LPA – ₹8.0 LPA **Mid-Level Python Engineer (1–3 Yrs):** ₹8.5 LPA – ₹16.5 LPA **Senior Backend / AI Specialist (3–6+ Yrs):** ₹18.0 LPA – ₹34.0+ LPA

MockAttempt Academy Pvt. Ltd. • Email: support@mockattempt.com • Phone/WhatsApp: +91 98765 43210 • Web: https://mockattempt.com/courses/python-training

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