Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 95400 07917
Course Code: PBCP • **Updated:** 19 Nov 2026
https://mockattempt.com/training/python-backend-developer-career-program

PBCP • PYTHON BACKEND, DJANGO & FASTAPI ENGINEERING **BEGINNER TO JOB-READY**

100% PLACEMENT GUARANTEED* CAREER PROGRAM (6-MO REFUND*)

Python Backend Developer – Industry-Ready Career Program

Python Backend Developer Career Program — Master Core & Advanced Python 3.12, OOP, 120+ DSA, PostgreSQL/SQL, SQLAlchemy 2.0, Django & DRF, FastAPI, JWT Security, Redis, Celery, RabbitMQ/Kafka, Pytest & Docker across 24 weekly modules and 6 mandatory projects.

Duration: 24 Weeks (6 Months) • Beginner to Job-Ready (600 Hrs) • **Modules:** 24 • **Live Topics:** 610 • **Subtopics:** 4,880+ • **Labs & Tasks:** 1220 • **Mocks / AI Vivas:** 610 / 610 • **Projects:** 24 Module + 1 Live Client Prod

All-Inclusive Direct Fee (Incl. 18% GST • Zero Hidden Charges): Total Payable: ₹15,000 (Net Tuition ₹12,712 + Included 18% GST ₹2,288) • **Split Option:** 3 × ₹5,000 (0% Installments) • **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 Career Live 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) • 28 Weeks (600 Guided Live + Lab Hours))

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

✓ **6-Month Placement Guarantee* + Online AI Assessment (OA) Support: Complete 100% assigned repositories & score ≥90% across all Topic Mock Tests & Voice AI Interviews → 100+ verified IT job openings within 6 months with active Senior Trainer coaching to crack employer Online AI Assessments (OA), or 100% net tuition refund*.**

Core Technologies & Engineering Stack: Python 3.12 (Core & Advanced) • Python OOP & Dataclasses • Decorators, Generators & Iterators • Threading, Multiprocessing & Asyncio • Python DSA (120+ Problems) • Git & GitHub PR Workflow • PostgreSQL 16 & Complete SQL • SQLite & MySQL Fundamentals • SQLAlchemy 2.0 & Alembic • MongoDB & PyMongo Fundamentals • Django 5 • Django REST Framework (DRF) • FastAPI & Pydantic v2 • REST APIs, Swagger & OpenAPI • JWT & OAuth 2.0 Fundamentals • Redis 7 Caching & Locks • Celery & Celery Beat • RabbitMQ & Apache Kafka Fundamentals • Pytest, unittest, Ruff & mypy • Docker & Docker Compose • Linux CLI & Nginx • Gunicorn & Uvicorn ASGI/WSGI • GitHub Actions CI/CD & AWS Fundamentals • pip, venv, Poetry & Postman

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Core & Advanced Python 3.12: Write clean, modular, type-annotated Python programs using collections, OOP, decorators, generators, context managers, and concurrency (`threading`, `multiprocessing`, `asyncio`).
2. Data Structures & Algorithms in Python: Solve 100–150 Easy and Medium coding problems across all 20 DSA categories with clear Big-O time & space justification.
3. PostgreSQL, SQL & SQLAlchemy 2.0: Design normalized 3NF relational databases, tune composite indexes with `EXPLAIN ANALYZE`, write complex joins/CTEs/window functions, and implement N+1-free SQLAlchemy 2.0 + Alembic persistence (plus MongoDB fundamentals).
4. Django 5 & Django REST Framework: Build, secure, document, cache, and test complete database-driven business backends with custom User models, serializers, ViewSets, filtering, pagination, and SimpleJWT RBAC.
5. FastAPI & Microservices Fundamentals: Develop high-performance asynchronous APIs with Pydantic v2, `Depends()` DI, Async SQLAlchemy, service boundaries, Saga compensation, Circuit Breakers, and `Idempotency-Key` safety.
6. Redis, Celery & Event Brokers: Implement Cache-Aside caching, distributed locks, Celery background/scheduled email & invoice jobs, RabbitMQ exchanges/DLQs, and Apache Kafka partitioned event streams.
7. Pytest, Linux, Docker, Nginx & CI/CD: Enforce `>= 80%` Pytest coverage + `Ruff`/`Black`/`mypy`, collaborate via GitHub PRs, containerize with non-root multi-stage Docker Compose + Nginx + Gunicorn/Uvicorn, automate GitHub Actions CI/CD, and defend complete backend systems in interviews.
8. Crack LeetCode and HackerRank problems in Python 3.12 & PostgreSQL SQL with live trainer mentorship on algorithmic/query patterns, edge cases, and optimal complexity.
9. Receive active Senior Trainer coaching during your 6-month (180-day) Placement Tenure to crack employer Online AI Assessments (OA), timed coding rounds, and automated screening tests.

10. Automated Testing, Debugging & Quality Gates (Python 3.12 (Core & Advanced), Python OOP & Dataclasses, Decorators, Generators & Iterators, Threading, Multiprocessing & Asyncio, Python DSA (120+ Problems)) — Automated Python 3.12 (Core & Advanced), Python OOP & Dataclasses, Decorators, Generators & Iterators, Threading, Multiprocessing & Asyncio, Python DSA (120+ Problems) test suites • Linting & static checks • Mentor Git pull-request reviews — ensuring your code never breaks silently when real users hit edge cases.
11. 100% Topic-Wise Timed Mock Tests, Spoken MockAttempt AI Viva & Live Client Production — Verify every single Python Backend Developer – Industry-Ready Career Program topic through timed MCQ exams and spoken technical interviews before deploying your Live Client Project.
12. Translated live client functional and non-functional requirements into modular Python 3.12 (Core & Advanced), Python OOP & Dataclasses, Decorators, Generators & Iterators, Threading, Multiprocessing & Asyncio, Python DSA (120+ Problems) production architecture.

Eligibility & Prerequisites: No prior Python, SQL, Django, or FastAPI experience required — Week 1 starts from programming fundamentals and Python 3.12 installation. • Consistent 25 hours/week commitment across live concept sessions, coding labs, GitHub assignments, and continuous Python DSA practice. • 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 (24 MODULES • 610 LIVE TOPICS • 4,880+ SUBTOPICS)

Progressive Zero-to-Expert Mastery Stages: Modules 1–4: Month 1 – Complete Core and Advanced Python (Weeks 1–4) (102 Topics / 816 Subtopics) • Modules 5–8: Month 2 – Advanced Python, DSA and Git (Weeks 5–8) (118 Topics / 944 Subtopics) • Modules 9–12: Month 3 – Database, SQL and ORM (Weeks 9–12) (96 Topics / 768 Subtopics) • Modules 13–16: Month 4 – Django and Django REST Framework (Weeks 13–16) (99 Topics / 792 Subtopics) • Modules 17–20: Month 5 – FastAPI, Microservices and Backend Engineering (Weeks 17–20) (101 Topics / 808 Subtopics) • Modules 21–24: Month 6 – Testing, Security, Docker, Deployment and Placement Preparation (Weeks 21–24) (94 Topics / 752 Subtopics)

MODULE 1 OF 24 • STAGE 1 • 25 LIVE + LAB HOURS • 22 TOPICS (176 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: Introduction to Python and Programming Fundamentals

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 Edge-Case & Pitfall Guard (Introduction to Programming) • Hands-On Python Implementation & Verification (Introduction to Programming) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Python Introduction) • Hands-On Python Implementation & Verification (Python Introduction) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Python Installation) • Hands-On Python Implementation & Verification (Python Installation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Environment Setup) • Hands-On Python Implementation & Verification (Environment Setup) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Python Execution) • Hands-On Python Implementation & Verification (Python Execution) • Python Execution — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Python Execution • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Python Syntax) • Hands-On Python Implementation & Verification (Python Syntax) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Variables) • Hands-On Python Implementation & Verification (Variables) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Data Types) • Hands-On Python Implementation & Verification (Data Types) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Type Conversion) • Hands-On Python Implementation & Verification (Type Conversion) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Input and Output) • Hands-On Python Implementation & Verification (Input and Output) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Operators) • Hands-On Python Implementation & Verification (Operators) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Bitwise Operators) • Hands-On Python Implementation & Verification (Bitwise Operators) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Membership Operators) • Hands-On Python Implementation & Verification (Membership Operators) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Identity Operators) • Hands-On Python Implementation & Verification (Identity Operators) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Operator Precedence) • Hands-On Python Implementation & Verification (Operator Precedence) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Conditional Statements) • Hands-On Python Implementation & Verification (Conditional Statements) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 → Next Topic 1.17: Nested Conditions

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

Subtopics Covered: Multi-criteria decision trees vs early-return guard clauses to flatten deeply indented nested `if` blocks • Production Edge-Case & Pitfall Guard (Nested Conditions) • Hands-On Python Implementation & Verification (Nested Conditions) • Nested Conditions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nested Conditions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Checking loan/job eligibility (active account -> verified email -> experience threshold) cleanly using flat guard clauses.

What Topic Is Next → Next Topic 1.18: Loops

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

Subtopics Covered: Definite iteration with `for` over iterables • Indefinite condition-driven `while` loops (including loop `else`) • Production Edge-Case & Pitfall Guard (Loops) • Hands-On Python Implementation & Verification (Loops) • Loops — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Loops • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Iterating batch records with `for` and running interactive retry/guessing loops with `while`.

What Topic Is Next → Next Topic 1.19: Loop Control Statements

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

Subtopics Covered: Exiting early with `break` • Skipping iterations with `continue` • Placeholder `pass` • `for ... else` search patterns • Production Edge-Case & Pitfall Guard (Loop Control Statements) • Hands-On Python Implementation & Verification (Loop Control Statements) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Loop Control Statements

Purpose: Terminating divisor checks immediately upon finding a factor in a `sqrt(n)` Prime Number Checker.

What Topic Is Next → Next Topic 1.20: Range Function

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

Subtopics Covered: `range(stop)` • `range(start • Stop)` • Production Edge-Case & Pitfall Guard (Range Function) • Hands-On Python Implementation & Verification (Range Function) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Range Function • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Generating batch offsets `range(0, total, batch_size)` in `O(1)` memory without allocating million-element lists.

What Topic Is Next → Next Topic 1.21: Pattern Programming

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

Subtopics Covered: Nested row/column loop invariants for right triangles • Pyramids • Floyd’s number triangle • Character patterns • Production Edge-Case & Pitfall Guard (Pattern Programming) • Hands-On Python Implementation & Verification (Pattern Programming) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pattern Programming

Purpose: Sharpening nested-loop index math (`row`, `spaces`, `columns`) tested in fresher coding rounds.

What Topic Is Next → Next Topic 1.22: Basic Debugging

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

Subtopics Covered: Reading Python tracebacks bottom-up • Distinguishing `SyntaxError` vs `NameError`/`TypeError`/`ValueError`/`ZeroDivisionError` • Strategic state inspection • Production Edge-Case & Pitfall Guard (Basic Debugging) • Hands-On Python Implementation & Verification (Basic Debugging) • Basic Debugging — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Basic Debugging

Purpose: Pinpointing exact file, line number, and offending input values from production stack traces in seconds.

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 24 • STAGE 1 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Python Strings, Lists, Tuples, Sets and Dictionaries

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 Edge-Case & Pitfall Guard (Strings) • Hands-On Python Implementation & Verification (Strings) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (String Slicing) • Hands-On Python Implementation & Verification (String Slicing) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (String Methods) • Hands-On Python Implementation & Verification (String Methods) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (String Searching) • Hands-On Python Implementation & Verification (String Searching) • String Searching — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for String Searching • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (String Validation) • Hands-On Python Implementation & Verification (String Validation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (String Formatting) • Hands-On Python Implementation & Verification (String Formatting) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (String Immutability) • Hands-On Python Implementation & Verification (String Immutability) • String Immutability — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for String Immutability • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Lists) • Hands-On Python Implementation & Verification (Lists) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (List Operations) • Hands-On Python Implementation & Verification (List Operations) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (List Methods) • Hands-On Python Implementation & Verification (List Methods) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (List Comprehension) • Hands-On Python Implementation & Verification (List Comprehension) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Tuples) • Hands-On Python Implementation & Verification (Tuples) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Tuple Methods) • Hands-On Python Implementation & Verification (Tuple Methods) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Sets) • Hands-On Python Implementation & Verification (Sets) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Set Operations) • Hands-On Python Implementation & Verification (Set Operations) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Set Methods) • Hands-On Python Implementation & Verification (Set Methods) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Dictionaries) • Hands-On Python Implementation & Verification (Dictionaries) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Dictionary Methods) • Hands-On Python Implementation & Verification (Dictionary Methods) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python

Backend Developer – Industry-Ready Career Program Production Runtime 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 → Next Topic 2.19: Dictionary Comprehension

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

Subtopics Covered: `{k_expr: v_expr for item in iterable if cond}` • Inverting unique maps • Filtering sensitive keys from logs • Production Edge-Case & Pitfall Guard (Dictionary Comprehension) • Hands-On Python Implementation & Verification (Dictionary Comprehension) • Dictionary Comprehension — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dictionary Comprehension

Purpose: Redacting `password`/`token` keys from request dictionaries before writing structured audit logs.

What Topic Is Next → Next Topic 2.20: Nested Collections

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

Subtopics Covered: Lists of dictionaries • Dictionaries of lists/dicts • Safe chained `.get()` navigation • Grouping records by category/department • Production Edge-Case & Pitfall Guard (Nested Collections) • Hands-On Python Implementation & Verification (Nested Collections) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nested Collections

Purpose: Grouping flat SQL/API employee records by department and summarizing total payroll per department.

What Topic Is Next → Next Topic 2.21: Mutable and Immutable Objects

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

Subtopics Covered: Immutable (`int` • `float` • `bool` • `str` • Production Edge-Case & Pitfall Guard (Mutable and Immutable Objects) • Hands-On Python Implementation & Verification (Mutable and Immutable Objects) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mutable

Purpose: Preventing unintended caller mutation when passing list/dict payloads into helper functions.

What Topic Is Next → Next Topic 2.22: Shallow Copy

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

Subtopics Covered: `list.copy()` • `dict.copy()` • `s[:]` • `copy.copy()` — duplicating the outer container while sharing nested inner object references • Production Edge-Case & Pitfall Guard (Shallow Copy) • Hands-On Python Implementation & Verification (Shallow Copy) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Shallow Copy

Purpose: Cloning flat 1-level configuration dictionaries quickly without copying immutable string/int values.

What Topic Is Next → Next Topic 2.23: Deep Copy

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

Subtopics Covered: `copy.deepcopy()` • Recursive cloning of nested graphs • Memoization of cyclic references • Memory/CPU trade-offs • Production Edge-Case & Pitfall Guard (Deep Copy) • Hands-On Python Implementation & Verification (Deep Copy) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deep Copy

Purpose: Generating isolated per-request test fixtures from a deeply nested OpenAPI JSON payload template.

What Topic Is Next → Next Topic 2.24: Collection Use Cases

2.24 Collection Use Cases [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Decision matrix: `list` (ordered mutable sequence) • `tuple` (immutable fixed record) • `set` (`O(1)` unique membership) • `dict` (`O(1)` keyed lookup) • Production Edge-Case & Pitfall Guard (Collection Use Cases) • Hands-On Python Implementation & Verification (Collection Use Cases) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Collection Use Cases

Purpose: Replacing an `O(n * m)` nested list scan with an `O(n + m)` dictionary/set join in backend batch jobs.

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 24 • STAGE 1 • 25 LIVE + LAB HOURS • 25 TOPICS (200 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Python Functions, Modules and 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 Edge-Case & Pitfall Guard (Functions) • Hands-On Python Implementation & Verification (Functions) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Function Parameters) • Hands-On Python Implementation & Verification (Function Parameters) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Default Parameters) • Hands-On Python Implementation & Verification (Default Parameters) • Default Parameters — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Default Parameters • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Variable Arguments) • Hands-On Python Implementation & Verification (Variable Arguments) • Variable Arguments — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Return Statements) • Hands-On Python Implementation & Verification (Return Statements) • Return Statements — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Variable Scope (LEGB)) • Hands-On Python Implementation & Verification (Variable Scope (LEGB)) • Variable Scope (LEGB) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Lambda Functions) • Hands-On Python Implementation & Verification (Lambda Functions) • Lambda Functions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lambda Functions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Recursion) • Hands-On Python Implementation & Verification (Recursion) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Higher-Order Functions) • Hands-On Python Implementation & Verification (Higher-Order Functions) • Higher-Order Functions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Higher-Order Functions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (map() Function) • Hands-On Python Implementation & Verification (map() Function) • map() Function — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for map() Function • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (filter() Function) • Hands-On Python Implementation & Verification (filter() Function) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (functools.reduce()) • Hands-On Python Implementation & Verification (functools.reduce()) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Python Modules) • Hands-On Python Implementation & Verification (Python Modules) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Python Packages) • Hands-On Python Implementation & Verification (Python Packages) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Import System) • Hands-On Python Implementation & Verification (Import System) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Main Execution Guard (__name__)) • Hands-On Python Implementation & Verification (Main Execution Guard (__name__)) • Main Execution Guard (__name__) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Main Execution Guard (__name__) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Python Standard Library) • Hands-On Python Implementation & Verification (Python Standard Library) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Exception Handling) • Hands-On Python Implementation & Verification (Exception Handling) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 → Next Topic 3.19: Multiple Exceptions

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

Subtopics Covered: Handling distinct failure modes with separate `except` clauses vs grouped tuples `except (ConnectionError • TimeoutError) as exc:` • Production Edge-Case & Pitfall Guard (Multiple Exceptions) • Hands-On Python Implementation & Verification (Multiple Exceptions) • Multiple Exceptions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multiple Exceptions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Distinguishing `400 Bad Request` (`ValueError`) from `404 Not Found` (`LookupError`) and `503 Retryable` (`TimeoutError`).

What Topic Is Next → Next Topic 3.20: else and finally Clauses

3.20 else and finally Clauses [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why `try ... except ... else ... finally` keeps the `try` scope narrow (`else` runs only if no exception • `finally` runs ALWAYS) • Production Edge-Case & Pitfall Guard (else and finally Clauses) • Hands-On Python Implementation & Verification (else and finally Clauses) • else and finally Clauses — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for else • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Placing only the risky parse/I/O line inside `try`, running follow-up success logic in `else`, and releasing locks/timers in `finally`.

What Topic Is Next → Next Topic 3.21: Raising Exceptions (raise)

3.21 Raising Exceptions (raise) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Validating invariants with `raise ValueError(...)` • Re-raising active exceptions via bare `raise` • Chaining `raise DomainError(...) from exc` • Production Edge-Case & Pitfall Guard (Raising Exceptions (raise)) • Hands-On Python Implementation & Verification (Raising Exceptions (raise)) • Raising Exceptions (raise) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Raising Exceptions (raise)

Purpose: Translating low-level driver/JSON errors into clean service-layer exceptions while preserving original tracebacks via `from exc`.

What Topic Is Next → Next Topic 3.22: Custom Exceptions

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

Subtopics Covered: Designing a project base exception (`BankingError(Exception)`) • Rich domain subclasses (`InsufficientFundsError` • `AccountNotFoundError`) with attributes • Production Edge-Case & Pitfall Guard (Custom Exceptions) • Hands-On Python Implementation & Verification (Custom Exceptions) • Custom Exceptions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Exceptions

Purpose: Mapping domain exceptions (`DuplicateEmailError` -> HTTP 409, `InsufficientBalanceError` -> HTTP 422) cleanly in Django/FastAPI global handlers.

What Topic Is Next → Next Topic 3.23: Assertions

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

Subtopics Covered: `assert condition • Message` for internal developer invariants vs explicit `if ...: raise ValueError(...)` for runtime user validation • Production Edge-Case & Pitfall Guard (Assertions) • Hands-On Python Implementation & Verification (Assertions) • Assertions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Assertions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Documenting internal algorithm post-conditions during development and test runs.

What Topic Is Next → Next Topic 3.24: Debugging with pdb

3.24 Debugging with pdb [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Python 3.7+ `breakpoint()` • `pdb` commands (`n` next • `s` step • `c` continue • Production Edge-Case & Pitfall Guard (Debugging with pdb) • Hands-On Python Implementation & Verification (Debugging with pdb) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Debugging with pdb

Purpose: Stepping frame-by-frame through a failing serializer or pricing calculation inside Docker or a remote terminal.

What Topic Is Next → Next Topic 3.25: Application Logging

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

Subtopics Covered: `logging` module (`DEBUG` • `INFO` • `WARNING` • `ERROR` • Production Edge-Case & Pitfall Guard (Application Logging) • Hands-On Python Implementation & Verification (Application Logging) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Application Logging

Purpose: Recording structured timestamped audit trails and full exception tracebacks (`logger.exception(...)`) without leaking secrets.

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 24 • STAGE 1 • 25 LIVE + LAB HOURS • 31 TOPICS (248 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: Complete Python Object-Oriented Programming and File Handling

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 Edge-Case & Pitfall Guard (OOP Introduction) • Hands-On Python Implementation & Verification (OOP Introduction) • OOP Introduction — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Classes) • Hands-On Python Implementation & Verification (Classes) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Objects) • Hands-On Python Implementation & Verification (Objects) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Constructors (__init__)) • Hands-On Python Implementation & Verification (Constructors (__init__)) • Constructors (__init__) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (self Instance Reference) • Hands-On Python Implementation & Verification (self Instance Reference) • self Instance Reference — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Instance Variables) • Hands-On Python Implementation & Verification (Instance Variables) • Instance Variables — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Instance Variables • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Class Variables) • Hands-On Python Implementation & Verification (Class Variables) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Instance Methods) • Hands-On Python Implementation & Verification (Instance Methods) • Instance Methods — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Class Methods (@classmethod)) • Hands-On Python Implementation & Verification (Class Methods (@classmethod)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Static Methods (@staticmethod)) • Hands-On Python Implementation & Verification (Static Methods (@staticmethod)) • Static Methods (@staticmethod) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Static Methods (@staticmethod) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

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 Edge-Case & Pitfall Guard (Encapsulation) • Hands-On Python Implementation & Verification (Encapsulation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Single Inheritance) • Hands-On Python Implementation & Verification (Single Inheritance) • Single Inheritance — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Multiple Inheritance) • Hands-On Python Implementation & Verification (Multiple Inheritance) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Multilevel Inheritance) • Hands-On Python Implementation & Verification (Multilevel Inheritance) • Multilevel Inheritance — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multilevel Inheritance • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Hierarchical Inheritance) • Hands-On Python Implementation & Verification (Hierarchical Inheritance) • Hierarchical Inheritance — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Polymorphism and Duck Typing) • Hands-On Python Implementation & Verification (Polymorphism and Duck Typing) • Polymorphism and Duck Typing — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Abstraction) • Hands-On Python Implementation & Verification (Abstraction) • Abstraction — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (abc Module (Abstract Base Classes)) • Hands-On Python Implementation & Verification (abc Module (Abstract Base Classes)) • abc Module (Abstract Base Classes) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 → Next Topic 4.19: Composition

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

Subtopics Covered: Strong `has-a` ownership (`Order` owns `OrderLine` items • `EmployeeService` composes a `Repository` + `Logger`) — "favor composition over inheritance" • Production Edge-Case & Pitfall Guard (Composition) • Hands-On Python Implementation & Verification (Composition) • Composition — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Composition • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Injecting a `Repository` and `Validator` into `EmployeeService` so storage can be mocked effortlessly in `pytest`.

What Topic Is Next → Next Topic 4.20: Aggregation

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

Subtopics Covered: Loose `has-a` relationship where child objects (`Employee`) have an independent lifecycle from the container (`Department`) • Production Edge-Case & Pitfall Guard (Aggregation) • Hands-On Python Implementation & Verification (Aggregation) • Aggregation — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Aggregation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Modeling `Department` aggregating `Employee` references — dissolving a project team does NOT delete the employees!

What Topic Is Next → Next Topic 4.21: Method Resolution Order (MRO)

4.21 Method Resolution Order (MRO) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: C3 Linearization algorithm • Inspecting `ClassName.mro()` / `__mro__` • How `super()` follows MRO in diamond hierarchies • Production Edge-Case & Pitfall Guard (Method Resolution Order (MRO)) • Hands-On Python Implementation & Verification (Method Resolution Order (MRO)) • Method Resolution Order (MRO) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Method Resolution Order (MRO)

Purpose: Predicting exact method execution order in Django Class-Based Views and DRF Mixin stacks (`CreateModelMixin`, `GenericAPIView`).

What Topic Is Next → Next Topic 4.22: Magic / Dunder Methods (__str__, __repr__, __len__, __eq__)

4.22 Magic / Dunder Methods (__str__, __repr__, __len__, __eq__) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unambiguous developer `__repr__` vs user-friendly `__str__` • Container `__len__` • Value equality `__eq__` (returning `NotImplemented` on foreign types) • `__lt__` • Production Edge-Case & Pitfall Guard (Magic / Dunder Methods (__str__, __repr__, __len__, __eq__)) • Hands-On Python Implementation & Verification (Magic / Dunder Methods (__str__, __repr__, __len__, __eq__)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Magic / Dunder Methods (__str__, __repr__, __len__,...

Purpose: Making domain objects print informative `Employee(emp_id='E1', name='Aarav')` logs instead of `<Employee object at 0x7f...>`.

What Topic Is Next → Next Topic 4.23: Properties (@property and Setters)

4.23 Properties (@property and Setters) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@property` computed getters • `@attr.setter` validation mutators • Read-only attributes without breaking `obj.attr` caller syntax • Production Edge-Case & Pitfall Guard (Properties (@property and Setters)) • Hands-On Python Implementation & Verification (Properties (@property and Setters)) • Properties (@property and Setters) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Properties (@property and Setters)

Purpose: Adding non-negative salary validation retroactively via `@salary.setter` without breaking existing `emp.salary = 80000` callers!

What Topic Is Next → Next Topic 4.24: Dataclasses (@dataclass)

4.24 Dataclasses (@dataclass) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from dataclasses import dataclass • Asdict` • Auto-generated `__init__` • `__repr__` • Production Edge-Case & Pitfall Guard (Dataclasses (@dataclass)) • Hands-On Python Implementation & Verification (Dataclasses (@dataclass)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dataclasses (@dataclass)

Purpose: Modeling concise, type-hinted DTOs and `Employee` records serializable directly via `asdict(emp)` to JSON.

What Topic Is Next → Next Topic 4.25: File Handling (open, read, write, append)

4.25 File Handling (open, read, write, append) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `open(path • Mode • Encoding="utf-8")` • `.read()` • Production Edge-Case & Pitfall Guard (File Handling (open, read, write, append)) • Hands-On Python Implementation & Verification (File Handling (open, read, write, append)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts &

non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File Handling (open, read, write, append)

Purpose: Streaming multi-gigabyte server access logs line-by-line (`for line in f:`) in `O(1)` RAM instead of `.read()` OOM crashes.

What Topic Is Next → Next Topic 4.26: File Modes (r, w, a, x and Binary)

4.26 File Modes (r, w, a, x and Binary) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read `r` • Truncate-write `w` • Append `a` • Exclusive-create `x` (`FileExistsError`) • Production Edge-Case & Pitfall Guard (File Modes (r, w, a, x and Binary)) • Hands-On Python Implementation & Verification (File Modes (r, w, a, x and Binary)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for File Modes (r, w, a, x and Binary)

Purpose: Using `wb` when saving uploaded PDF resumes and `a` when appending audit log entries without wiping history.

What Topic Is Next → Next Topic 4.27: Context Managers (with Statement)

4.27 Context Managers (with Statement) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `with open(...) as f:` deterministic `__enter__` / `__exit__` cleanup even when exceptions are raised mid-block • Production Edge-Case & Pitfall Guard (Context Managers (with Statement)) • Hands-On Python Implementation & Verification (Context Managers (with Statement)) • Context Managers (with Statement) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Context Managers (with Statement) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Preventing file-descriptor exhaustion (`OSError: Too many open files`) in long-running backend workers.

What Topic Is Next → Next Topic 4.28: CSV Reading and Writing

4.28 CSV Reading and Writing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `csv.reader` • `csv.writer` • `csv.DictReader` • `csv.DictWriter` • Production Edge-Case & Pitfall Guard (CSV Reading and Writing) • Hands-On Python Implementation & Verification (CSV Reading and Writing) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CSV Reading

Purpose: Importing bulk HR onboarding spreadsheets and exporting monthly payroll CSV reports with proper comma/quote escaping.

What Topic Is Next → Next Topic 4.29: JSON Serialization and Deserialization

4.29 JSON Serialization and Deserialization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `json.dumps()` / `json.dump(... • Indent=2)` • `json.loads()` / `json.load()` • `json.JSONDecodeError` • Production Edge-Case & Pitfall Guard (JSON Serialization and Deserialization) • Hands-On Python Implementation & Verification (JSON Serialization and Deserialization) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JSON Serialization

Purpose: Persisting structured Employee records to `employees.json` in Mini-Project 1 and exchanging REST API payloads.

What Topic Is Next → Next Topic 4.30: Pickle Object Serialization Fundamentals

4.30 Pickle Object Serialization Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Binary Python object serialization (`pickle.dumps`/`loads`) • Python-only format limitations • Arbitrary code execution security risks • Production Edge-Case & Pitfall Guard (Pickle Object Serialization Fundamentals) • Hands-On Python Implementation & Verification (Pickle Object Serialization Fundamentals) • Pickle Object Serialization Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pickle Object Serialization Fundamentals

Purpose: Understanding internal Celery/cache serialization options and why JSON/MsgPack are strongly preferred over `pickle` for external data.

What Topic Is Next → Next Topic 4.31: pathlib File and Path Management

4.31 pathlib File and Path Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from pathlib import Path` • `/` path joining • `.resolve()` • `.exists()` • Production Edge-Case & Pitfall Guard (pathlib File and Path Management) • Hands-On Python Implementation & Verification (pathlib File and Path Management) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for pathlib File

Purpose: Writing OS-independent file storage paths (`BASE_DIR / "data" / "employees.json"`) that work identically on Windows `\` and Linux `/`!

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 24 • STAGE 2 • 25 LIVE + LAB HOURS • 29 TOPICS (232 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Advanced Python Programming

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 Edge-Case & Pitfall Guard (Iterables) • Hands-On Python Implementation & Verification (Iterables) • Iterables — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Iterators) • Hands-On Python Implementation & Verification (Iterators) • Iterators — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Iterators • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

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 Edge-Case & Pitfall Guard (iter() and next()) • Hands-On Python Implementation & Verification (iter() and next()) • iter() and next() — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Generators (yield)) • Hands-On Python Implementation & Verification (Generators (yield)) • Generators (yield) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Generator Expressions) • Hands-On Python Implementation & Verification (Generator Expressions) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Function Decorators) • Hands-On Python Implementation & Verification (Function Decorators) • Function Decorators — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Function Decorators • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Nested Decorators) • Hands-On Python Implementation & Verification (Nested Decorators) • Nested Decorators — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nested Decorators • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Closures) • Hands-On Python Implementation & Verification (Closures) • Closures — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Closures • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Custom Context Managers (__enter__ and __exit__)) • Hands-On Python Implementation & Verification (Custom Context Managers (__enter__ and __exit__)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (contextlib (@contextmanager)) • Hands-On Python Implementation & Verification (contextlib (@contextmanager)) • contextlib (@contextmanager) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Advanced Comprehensions) • Hands-On Python Implementation & Verification (Advanced Comprehensions) • Advanced Comprehensions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Extended Iterable Unpacking) • Hands-On Python Implementation & Verification (Extended Iterable Unpacking) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Walrus Operator (:=)) • Hands-On Python Implementation & Verification (Walrus Operator (:=)) • Walrus Operator (:=) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Walrus Operator (:=) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Structural Pattern Matching (match and case)) • Hands-On Python Implementation & Verification (Structural Pattern Matching (match and case)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Type Hints and Annotations) • Hands-On Python Implementation & Verification (Type Hints and Annotations) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (typing Module (List, Dict, Optional, Union, Callable)) • Hands-On Python Implementation & Verification (typing Module (List, Dict, Optional, Union, Callable)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 → Next Topic 5.17: Generic Typing (TypeVar and Generic)

5.17 Generic Typing (TypeVar and Generic) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Python 3.12 PEP 695 generic syntax `class Page[T]:` • `def first[T](items: list[T]) -> T:` (plus `TypeVar`/`Generic[T]`) • Production Edge-Case & Pitfall Guard (Generic Typing (TypeVar and Generic)) • Hands-On Python Implementation & Verification (Generic Typing (TypeVar and Generic)) • Generic Typing (TypeVar and Generic) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Generic Typing (TypeVar and

Generic) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Building reusable `PaginatedResponse[T]` and `BaseRepository[ModelT]` wrappers across FastAPI endpoints.

What Topic Is Next → Next Topic 5.18: Protocols (Structural Subtyping)

5.18 Protocols (Structural Subtyping) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from typing import Protocol • Runtime_checkable` — static duck typing without forcing nominal `ABC` inheritance • Production Edge-Case & Pitfall Guard (Protocols (Structural Subtyping)) • Hands-On Python Implementation & Verification (Protocols (Structural Subtyping)) • Protocols (Structural Subtyping) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Protocols (Structural Subtyping) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Typing `Notifier(Protocol)` so fake test doubles in `pytest` satisfy type checkers without inheriting production base classes.

What Topic Is Next → Next Topic 5.19: Advanced Dataclasses (field, frozen, slots)

5.19 Advanced Dataclasses (field, frozen, slots) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@dataclass(frozen=True • Slots=True • Kw_only=True)` • `field(default_factory=... • Production Edge-Case & Pitfall Guard (Advanced Dataclasses (field, frozen, slots)) • Hands-On Python Implementation & Verification (Advanced Dataclasses (field, frozen, slots)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Advanced Dataclasses (field, frozen, slots)

Purpose: Creating hashable, memory-compact (`slots=True`), immutable value objects (`Money`, `JwtClaims`) hiding secrets via `repr=False`.

What Topic Is Next → Next Topic 5.20: Enums (Enum and IntEnum)

5.20 Enums (Enum and IntEnum) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from enum import Enum • IntEnum • StrEnum • Auto` for type-safe finite states (`ApplicationStatus` • Production Edge-Case & Pitfall Guard (Enums (Enum and IntEnum)) • Hands-On Python Implementation & Verification (Enums (Enum and IntEnum)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Enums (Enum and IntEnum)

Purpose: Using `StrEnum` in Python 3.12 for Django/FastAPI status columns so values serialize cleanly as JSON strings without magic typos.

What Topic Is Next → Next Topic 5.21: collections Module (Counter, defaultdict, namedtuple, deque)

5.21 collections Module (Counter, defaultdict, namedtuple, deque) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Counter.most_common()` • `defaultdict(list)` grouping • `namedtuple` lightweight records • `deque(maxlen=N)` `O(1)` double-ended queues • Production Edge-Case & Pitfall Guard (collections Module (Counter, defaultdict, namedtuple, deque)) • Hands-On Python Implementation & Verification (collections Module (Counter, defaultdict, namedtuple, deque)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for collections Module (Counter, defaultdict, namedtuple...

Purpose: Using `deque` for `O(1)` BFS queues/sliding-window rate logs and `Counter` for analytics aggregation.

What Topic Is Next → Next Topic 5.22: functools (partial, lru_cache, wraps)

5.22 functools (partial, lru_cache, wraps) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@lru_cache(maxsize=256)` / `@cache` memoization • `cache_info()` • `partial()` argument pre-binding • `@wraps` • Production Edge-Case & Pitfall Guard (functools (partial, lru_cache, wraps)) • Hands-On Python Implementation & Verification (functools (partial, lru_cache, wraps)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for functools (partial, lru_cache, wraps)

Purpose: Caching parsed Pydantic settings (`@lru_cache def get_settings()`) in FastAPI so `.env` is read once per worker.

What Topic Is Next → Next Topic 5.23: itertools (chain, combinations, permutations, groupby)

5.23 itertools (chain, combinations, permutations, groupby) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `itertools.chain.from_iterable` • `islice` • `batched` (Python 3.12!) • `combinations` • Production Edge-Case & Pitfall Guard (itertools (chain, combinations, permutations, groupby)) • Hands-On Python Implementation & Verification (itertools (chain, combinations, permutations, groupby)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for itertools (chain, combinations, permutations, groupb...

Purpose: Chunking large streams with Python 3.12 `itertools.batched(records, 500)` for bulk SQLAlchemy inserts.

What Topic Is Next → Next Topic 5.24: Regular Expressions (re Module)

5.24 Regular Expressions (re Module) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `re.compile()` • Raw patterns `r"..."` • `search` vs `match` vs `fullmatch` • `findall` • Production Edge-Case & Pitfall Guard (Regular Expressions (re Module)) • Hands-On Python Implementation & Verification (Regular Expressions (re Module)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Regular Expressions (re Module)

Purpose: Validating SKU/PAN/slug formats with `re.fullmatch()` and redacting phone/Aadhaar patterns from logs via `re.sub()`.

What Topic Is Next → Next Topic 5.25: Environment Variables (os.environ)

5.25 Environment Variables (os.environ) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `os.environ.get("KEY" • Default)` vs mandatory `os.environ["SECRET_KEY"]` fail-fast startup validation • `.env` loading via `python-dotenv` • Production Edge-Case & Pitfall Guard (Environment Variables (os.environ)) • Hands-On Python Implementation & Verification (Environment Variables (os.environ)) • Environment Variables (os.environ) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment Variables (os.environ)

Purpose: Injecting `DATABASE_URL`, `REDIS_URL`, and `JWT_SECRET_KEY` via environment variables (12-Factor App principle) with zero secrets in Git.

What Topic Is Next → Next Topic 5.26: Application Configuration Management

5.26 Application Configuration Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Layered `BaseConfig` -> `DevConfig` / `TestConfig` / `ProdConfig` dataclasses • `.env.example` templates • Feature flags • Production Edge-Case & Pitfall Guard (Application Configuration Management) • Hands-On Python Implementation & Verification (Application Configuration Management) • Application Configuration Management — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Application Configuration Management

Purpose: Switching cleanly between local SQLite/PostgreSQL dev databases, isolated test DBs, and production RDS endpoints.

What Topic Is Next → Next Topic 5.27: Python Packaging (pyproject.toml)

5.27 Python Packaging (pyproject.toml) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern PEP 621 `pyproject.toml` metadata • `src/` vs flat layout • Editable installs `pip install -e .` • Tool config (`[tool.pytest]` • Production Edge-Case & Pitfall Guard (Python Packaging (pyproject.toml)) • Hands-On Python Implementation & Verification (Python Packaging (pyproject.toml)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Python Packaging (pyproject.toml)

Purpose: Centralizing project dependencies, `pytest` paths, and `ruff`/`mypy` settings inside one standard `pyproject.toml` file.

What Topic Is Next → Next Topic 5.28: Virtual Environments (venv)

5.28 Virtual Environments (venv) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `python -m venv .venv` • Activation on Windows (`.venv\Scripts\activate`) • Linux (`source .venv/bin/activate`) • Site-packages isolation • Production Edge-Case & Pitfall Guard (Virtual Environments (venv)) • Hands-On Python Implementation & Verification (Virtual Environments (venv)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Virtual Environments (venv)

Purpose: Isolating Django 5 project dependencies from FastAPI microservice dependencies on the same developer workstation.

What Topic Is Next → Next Topic 5.29: Dependency Management (pip, requirements.txt and Poetry)

5.29 Dependency Management (pip, requirements.txt and Poetry) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pinned deterministic `requirements.txt` vs `requirements-dev.txt` • Semantic version constraints • `Poetry` (`poetry.lock`) fundamentals • Production Edge-Case & Pitfall Guard (Dependency Management (pip, requirements.txt and Poetry)) • Hands-On Python Implementation & Verification (Dependency Management (pip, requirements.txt and Poetry)) • Dependency Management (pip, requirements.txt and Poetry) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dependency Management (pip, requirements.txt and Poe...

Purpose: Guaranteeing Docker builds and CI runners install the exact same tested library versions as local development.

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 24 • STAGE 2 • 25 LIVE + LAB HOURS • 30 TOPICS (240 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Python Memory Management, Concurrency and Asynchronous Programming

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 Edge-Case & Pitfall Guard (Python Execution Model) • Hands-On Python Implementation & Verification (Python Execution Model) • Python Execution Model — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Python Execution Model • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

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 Edge-Case & Pitfall Guard (CPython Runtime Architecture) • Hands-On Python Implementation & Verification (CPython Runtime Architecture) • CPython Runtime Architecture — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Python Bytecode Compilation (dis)) • Hands-On Python Implementation & Verification (Python Bytecode Compilation (dis)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Memory Allocation in Python) • Hands-On Python Implementation & Verification (Memory Allocation in Python) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Reference Counting) • Hands-On Python Implementation & Verification (Reference Counting) • Reference Counting — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Cyclic Garbage Collection (gc)) • Hands-On Python Implementation & Verification (Cyclic Garbage Collection (gc)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Mutable Object References in Memory) • Hands-On Python Implementation & Verification (Mutable Object References in Memory) • Mutable Object References in Memory — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Mutable Object References in Memory • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Shallow vs Deep Copying in Memory) • Hands-On Python Implementation & Verification (Shallow vs Deep Copying in Memory) • Shallow vs Deep Copying in Memory — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Global Interpreter Lock (GIL)) • Hands-On Python Implementation & Verification (Global Interpreter Lock (GIL)) • Global Interpreter Lock (GIL) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Threading Fundamentals) • Hands-On Python Implementation & Verification (Threading Fundamentals) • Threading Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Creating and Managing Threads) • Hands-On Python Implementation & Verification (Creating and Managing Threads) • Idiomatic PBCP Implementation: clean modular architecture, strict

contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Thread Synchronization (Lock and RLock)) • Hands-On Python Implementation & Verification (Thread Synchronization (Lock and RLock)) • Thread Synchronization (Lock and RLock) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Thread Synchronization (Lock and RLock) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Race Conditions) • Hands-On Python Implementation & Verification (Race Conditions) • Race Conditions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Race Conditions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 Edge-Case & Pitfall Guard (Deadlocks and Prevention) • Hands-On Python Implementation & Verification (Deadlocks and Prevention) • Deadlocks and Prevention — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Thread Pools (ThreadPoolExecutor)) • Hands-On Python Implementation & Verification (Thread Pools (ThreadPoolExecutor)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 Edge-Case & Pitfall Guard (Multiprocessing (Process-Based Parallelism)) • Hands-On Python Implementation & Verification (Multiprocessing (Process-Based Parallelism)) • Multiprocessing (Process-Based Parallelism) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 → Next Topic 6.17: Process Pools (ProcessPoolExecutor)

6.17 Process Pools (ProcessPoolExecutor) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `concurrent.futures.ProcessPoolExecutor` unified `Future` API • Pickling overhead vs chunk size (`chunksize`) • CPU core sizing • Production Edge-Case & Pitfall Guard (Process Pools (ProcessPoolExecutor)) • Hands-On Python Implementation & Verification (Process Pools (ProcessPoolExecutor)) • Process Pools (ProcessPoolExecutor) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Process Pools (ProcessPoolExecutor)

Purpose: Speeding up heavy batch data checksum/parsing jobs linearly with available CPU cores.

What Topic Is Next → Next Topic 6.18: Inter-Process Communication (Queue and Pipe)

6.18 Inter-Process Communication (Queue and Pipe) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why separate processes do NOT share global variables • Passing messages safely via `multiprocessing.Queue` • Two-endpoint `Pipe` • Production Edge-Case & Pitfall Guard (Inter-Process Communication (Queue and Pipe)) • Hands-On Python Implementation & Verification (Inter-Process Communication (Queue and Pipe)) • Inter-Process Communication (Queue and Pipe) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Inter-Process Communication (Queue and Pipe)

Purpose: Streaming parsed records from CPU worker processes back to a writer process via a bounded `Queue`.

What Topic Is Next → Next Topic 6.19: Threading vs Multiprocessing Comparison

6.19 Threading vs Multiprocessing Comparison [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Memory footprint • Startup latency • GIL impact • Crash isolation • Production Edge-Case & Pitfall Guard (Threading vs Multiprocessing Comparison) • Hands-On Python Implementation & Verification (Threading vs Multiprocessing Comparison) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Threading vs Multiprocessing Comparison

Purpose: Configuring Gunicorn `--workers` (processes for CPU cores) vs `--threads` / Uvicorn workers (for I/O concurrency).

What Topic Is Next → Next Topic 6.20: Asynchronous Programming Model

6.20 Asynchronous Programming Model [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cooperative single-threaded multitasking vs preemptive OS threads • Non-blocking sockets (`epoll`/`IOCP`) • Why blocking the loop freezes all requests • Production Edge-Case & Pitfall Guard (Asynchronous Programming Model) • Hands-On Python Implementation & Verification (Asynchronous Programming Model) • Asynchronous Programming Model — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Asynchronous Programming Model

Purpose: Serving thousands of concurrent FastAPI HTTP/WebSocket connections on a single Uvicorn event-loop thread.

What Topic Is Next → Next Topic 6.21: Coroutines with async and await

6.21 Coroutines with async and await [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `async def` coroutine functions • Coroutine objects (do nothing until awaited!) • `await` suspension points • Production Edge-Case & Pitfall Guard (Coroutines with async and await) • Hands-On Python Implementation & Verification (Coroutines with async and await) • Coroutines with async and await — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Coroutines with async

Purpose: Writing readable sequential-looking asynchronous FastAPI endpoints and `AsyncSession` database queries.

What Topic Is Next → Next Topic 6.22: Asyncio Event Loop

6.22 Asyncio Event Loop [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `asyncio.run()` lifecycle • Ready queue vs I/O selector callbacks • `asyncio.get_running_loop()` • `asyncio.to_thread()` for legacy sync calls • Production Edge-Case & Pitfall Guard (Asyncio Event Loop) • Hands-On Python Implementation & Verification (Asyncio Event Loop) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Asyncio Event Loop

Purpose: Offloading unavoidable synchronous SDK calls safely from an `async def` route via `await asyncio.to_thread(sync_fn, arg)`.

What Topic Is Next → Next Topic 6.23: asyncio Core Module

6.23 asyncio Core Module [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `asyncio.sleep` • `asyncio.wait_for(coro • Timeout=...)` • `asyncio.Semaphore` concurrency throttling • Production Edge-Case & Pitfall Guard (asyncio Core Module) • Hands-On Python Implementation & Verification (asyncio Core Module) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for asyncio Core Module

Purpose: Bounding outbound webhook fan-out to max 20 in-flight requests using `async with asyncio.Semaphore(20):` and `wait_for` timeouts.

What Topic Is Next → Next Topic 6.24: Concurrent Tasks (asyncio.create_task)

6.24 Concurrent Tasks (asyncio.create_task) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scheduling coroutines eagerly on the loop with `asyncio.create_task()` • Task cancellation (`task.cancel()`) • Python 3.11+ `asyncio.TaskGroup` • Production Edge-Case & Pitfall Guard (Concurrent Tasks (asyncio.create_task)) • Hands-On Python Implementation & Verification (Concurrent Tasks (asyncio.create_task)) • Concurrent Tasks (asyncio.create_task) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Concurrent Tasks (asyncio.create_task)

Purpose: Starting profile, applications, and recommendations queries concurrently via `TaskGroup` / `create_task` instead of sequential `await`s.

What Topic Is Next → Next Topic 6.25: Parallel Awaitables (asyncio.gather)

6.25 Parallel Awaitables (asyncio.gather) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `await asyncio.gather(*aws • Return_exceptions=True)` result ordering • Partial failure handling across concurrent calls • Production Edge-Case & Pitfall Guard (Parallel Awaitables (asyncio.gather)) • Hands-On Python Implementation & Verification (Parallel Awaitables (asyncio.gather)) • Parallel Awaitables (asyncio.gather) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Parallel Awaitables (asyncio.gather)

Purpose: Aggregating responses from 4 microservices where one optional recommendation failure (`return_exceptions=True`) degrades gracefully.

What Topic Is Next → Next Topic 6.26: Async Context Managers (async with)

6.26 Async Context Managers (async with) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `__aenter__` • `__aexit__` protocol • `@contextlib.asynccontextmanager` for async DB sessions • HTTP client pools • Production Edge-Case & Pitfall Guard (Async Context Managers (async with)) • Hands-On Python Implementation & Verification (Async Context Managers (async with)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Async Context Managers (async with)

Purpose: Managing `async with httpx.AsyncClient() as client:` and `async with async_session() as session:` cleanly.

What Topic Is Next → Next Topic 6.27: Async Iterators (__aiter__ and async for)

6.27 Async Iterators (__aiter__ and async for) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `__aiter__` • `async def __anext__` (`StopAsyncIteration`) consumed via `async for item in stream:` • Production Edge-Case & Pitfall Guard (Async Iterators (__aiter__ and async for)) • Hands-On Python Implementation & Verification (Async Iterators (__aiter__ and async for)) • Async Iterators (__aiter__ and async for) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Async Iterators (__aiter__ and async for) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Consuming incoming WebSocket frames, Kafka async messages, or Motor MongoDB cursors via `async for`.

What Topic Is Next → Next Topic 6.28: Async Generators

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

Subtopics Covered: `async def` containing `yield` • Async generator comprehensions • `contextlib.aclosing()` deterministic cleanup • Production Edge-Case & Pitfall Guard (Async Generators) • Hands-On Python Implementation & Verification (Async Generators) • Async Generators — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Async Generators

Purpose: Yielding Server-Sent Events (SSE) or FastAPI `Depends` `async def get_db(): yield session` dependencies!

What Topic Is Next → Next Topic 6.29: Async HTTP Requests (httpx)

6.29 Async HTTP Requests (httpx) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Using `httpx.AsyncClient(timeout=...)` • Connection pooling • `raise_for_status()` • Concurrent outbound REST API calls • Production Edge-Case & Pitfall Guard (Async HTTP Requests (httpx)) • Hands-On Python Implementation & Verification (Async HTTP Requests (httpx)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Async HTTP Requests (httpx)

Purpose: Calling Payment, Inventory, and Notification microservices non-blockingly from a FastAPI Order service.

What Topic Is Next → Next Topic 6.30: Choosing the Right Concurrency Model

6.30 Choosing the Right Concurrency Model [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Architecture decision matrix: `asyncio` vs `ThreadPoolExecutor` vs `ProcessPoolExecutor` vs Celery/Kafka distributed queues • Production Edge-Case & Pitfall Guard (Choosing the Right Concurrency Model) • Hands-On Python Implementation & Verification (Choosing the Right Concurrency Model) • Choosing the Right Concurrency Model — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Choosing the Right Concurrency Model • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Selecting `asyncio` for real-time API I/O, `ProcessPoolExecutor` for local CPU bursts, and Celery + Redis/RabbitMQ for durable background jobs.

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

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 24 • STAGE 2 • 25 LIVE + LAB HOURS • 30 TOPICS (240 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: Data Structures and Algorithms Using Python

Build strong algorithmic problem-solving foundations in Python 3.12 that continue across all 6 months (120 curated problems): Big-O time/space complexity, arrays, strings, dict/set hashing, two pointers, sliding window, prefix sums, `bisect` binary search, sorting, recursion, backtracking, linked lists, stacks, `deque` queues, binary trees, BSTs, traversals, `heapq` heaps, BFS/DFS graphs, greedy, dynamic programming, and bit manipulation.

Objectives: ✓ Analyze exact worst-case and amortized Time and Space Complexity (`O(1)`, `O(log n)`, `O(n)`, `O(n log n)`, `O(n^2)`) of Python code. • ✓ Solve array, string, hashing, two-pointer, sliding-window, prefix-sum, and `bisect` binary-search problems cleanly. • ✓ Implement and apply linked lists, monotonic stacks, `collections.deque` queues, binary trees, BSTs, and `heapq` priority queues. • ✓ Traverse graphs with BFS/DFS and apply recursion, backtracking, greedy choice, memoized/tabulated DP, and bitwise tricks. • ✓ Maintain continuous weekly progress toward the 100–150 Python DSA problem target in the MockAttempt Coding Lab.

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

Subtopics Covered: Why algorithmic efficiency matters in backend APIs (10ms vs 30s timeout on 100k rows) • Structured interview problem-solving • Production Edge-Case & Pitfall Guard (DSA Introduction) • Hands-On Python Implementation & Verification (DSA Introduction) • DSA Introduction — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DSA Introduction • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Preventing `O(n^2)` endpoint timeouts when matching thousands of job applications or inventory SKUs.

What Topic Is Next → Next Topic 7.2: Time Complexity Analysis

7.2 Time Complexity Analysis [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Counting dominant operations across single loops • Nested loops • Halving loops • Hidden Python library costs (`x in list` is `O(n)`!) • Production Edge-Case & Pitfall Guard (Time Complexity Analysis) • Hands-On Python Implementation & Verification (Time Complexity Analysis) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Time Complexity Analysis

Purpose: Spotting hidden `O(n^2)` bugs like `if item in seen_list:` inside a `for` loop (`O(n)` membership * `n` iterations!).

What Topic Is Next → Next Topic 7.3: Space Complexity Analysis

7.3 Space Complexity Analysis [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Auxiliary memory vs input memory • Recursion call-stack depth `O(h)` • Slice copy allocations `arr[i:j]` • In-place `O(1)` algorithms • Production Edge-Case & Pitfall Guard (Space Complexity Analysis) • Hands-On Python Implementation & Verification (Space Complexity Analysis) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Space Complexity Analysis

Purpose: Keeping auxiliary space `O(1)` or `O(k)` when processing large datasets in memory-constrained containers.

What Topic Is Next → Next Topic 7.4: Big O Notation (O(1), O(log n), O(n), O(n log n), O(n^2))

7.4 Big O Notation (O(1), O(log n), O(n), O(n log n), O(n^2)) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Asymptotic growth hierarchy • Dropping constants/lower-order terms • Best/average/worst cases • Amortized `O(1)` `list.append` • Production Edge-Case & Pitfall Guard (Big O Notation (O(1), O(log n), O(n), O(n log n), O(n^2))) • Hands-On Python Implementation & Verification (Big O Notation (O(1), O(log n), O(n), O(n log n), O(n^2))) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Big O Notation (O(1), O(log n), O(n), O(n log n), O(...

Purpose: Articulating crisp Time & Space trade-offs at the end of every coding interview round.

What Topic Is Next → Next Topic 7.5: Dynamic Arrays in Python Lists

7.5 Dynamic Arrays in Python Lists [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Over-allocated C pointer array backing Python `list` • `O(1)` indexing & tail `append`/`pop` • `O(n)` middle `insert`/`pop(i)` • Production Edge-Case & Pitfall Guard (Dynamic Arrays in Python Lists) • Hands-On Python Implementation & Verification (Dynamic Arrays in Python Lists) • Dynamic Arrays in Python Lists — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Arrays in Python Lists

Purpose: Implementing array rotation, Kadane's maximum subarray sum, and prefix/suffix product without division.

What Topic Is Next → Next Topic 7.6: Array Traversal and In-Place Manipulation

7.6 Array Traversal and In-Place Manipulation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single-pass read/write index pointers • Dutch National Flag 3-way partition (`0 • 2`) • Cyclic index placement • Production Edge-Case & Pitfall Guard (Array Traversal and In-Place Manipulation) • Hands-On Python Implementation & Verification (Array Traversal and In-Place Manipulation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Array Traversal

Purpose: Compacting, deduplicating, and partitioning buffers in-place without extra allocations.

What Topic Is Next → Next Topic 7.7: String Problem-Solving Patterns

7.7 String Problem-Solving Patterns [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Character frequency arrays (`ord(c) - ord("a")`) / `Counter` • Two-pointer palindrome checks • Anagram grouping • Center expansion • Production Edge-Case & Pitfall Guard (String Problem-Solving Patterns) • Hands-On Python Implementation & Verification (String Problem-Solving Patterns) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for String Problem-Solving Patterns

Purpose: Normalizing search queries, tokenizing slugs, and matching text patterns efficiently.

What Topic Is Next → Next Topic 7.8: Hashing with Python Dictionaries

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

Subtopics Covered: Complement lookup (`target - x` in `seen`) • Index mapping • Frequency counting • `OrderedDict` `O(1)` LRU cache design • Production Edge-Case & Pitfall Guard (Hashing with Python Dictionaries) • Hands-On Python Implementation & Verification (Hashing with Python Dictionaries) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Hashing with Python Dictionaries

Purpose: Turning `O(n^2)` pair/subarray searches into single-pass `O(n)` hash lookups and building `O(1)` LRU caches.

What Topic Is Next → Next Topic 7.9: Set-Based Lookup and Deduplication

7.9 Set-Based Lookup and Deduplication [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `O(1)` `set` existence checks • Sequence-start detection (`x - 1 not in num_set`) • Cycle/duplicate detection • Production Edge-Case & Pitfall Guard (Set-Based Lookup and Deduplication) • Hands-On Python Implementation & Verification (Set-Based Lookup and Deduplication) • Set-Based Lookup and Deduplication — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Set-Based Lookup

Purpose: Finding Longest Consecutive Sequence in `O(n)` without `O(n log n)` sorting.

What Topic Is Next → Next Topic 7.10: Two-Pointer Technique

7.10 Two-Pointer Technique [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Converging opposite-end pointers (`l = 0 • R = n - 1`) on sorted arrays • Container With Most Water • 3Sum • Production Edge-Case & Pitfall Guard (Two-Pointer Technique) • Hands-On Python Implementation & Verification (Two-Pointer Technique) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Two-Pointer Technique

Purpose: Solving sorted pair/triplet sum and optimal boundary problems in `O(1)` auxiliary space.

What Topic Is Next → Next Topic 7.11: Sliding Window (Fixed and Variable)

7.11 Sliding Window (Fixed and Variable) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fixed-size `k` rolling window • Variable `[l..r]` window expanding `r` • Shrinking `l` while invariant is violated • Production Edge-Case & Pitfall Guard (Sliding Window (Fixed and Variable)) • Hands-On Python Implementation & Verification (Sliding Window (Fixed and Variable)) • Sliding Window (Fixed and Variable) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Sliding Window (Fixed and Variable)

Purpose: Computing rolling 5-minute API error rates and Longest Substring Without Repeating Characters in `O(n)`.

What Topic Is Next → Next Topic 7.12: Prefix Sums and Range Queries

7.12 Prefix Sums and Range Queries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `pref[i + 1] = pref[i] + nums[i]` for `O(1)` range sums • Prefix-sum + hashmap (`pref - k`) subarray counting • Difference arrays • Production Edge-Case & Pitfall Guard (Prefix Sums and Range Queries) • Hands-On Python Implementation & Verification (Prefix Sums and Range Queries) • Prefix Sums and Range Queries — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Prefix Sums

Purpose: Answering instant date-range revenue queries and counting `Subarray Sum Equals K` (even with negative numbers!).

What Topic Is Next → Next Topic 7.13: Linear and Binary Search (bisect)

7.13 Linear and Binary Search (bisect) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Iterative binary search (`l <= r` vs lower-bound `l < r`) • `bisect_left` / `bisect_right` • Rotated sorted search • Binary search on answer space • Production Edge-Case & Pitfall Guard (Linear and Binary Search (bisect)) • Hands-On Python Implementation & Verification (Linear and Binary Search (bisect)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linear

Purpose: Locating boundary timestamps in sorted log arrays in `O(log n)` and minimizing ship/worker capacity via monotonic feasibility checks.

What Topic Is Next → Next Topic 7.14: Elementary Sorting (Bubble, Selection, Insertion)

7.14 Elementary Sorting (Bubble, Selection, Insertion) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: In-place comparison sorts • Stability definition • Why Insertion Sort runs in `O(n)` on nearly-sorted data • Production Edge-Case & Pitfall Guard (Elementary Sorting (Bubble, Selection, Insertion)) • Hands-On Python Implementation & Verification (Elementary Sorting (Bubble, Selection, Insertion)) • Elementary Sorting (Bubble, Selection, Insertion) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Elementary Sorting (Bubble, Selection, Insertion)

Purpose: Understanding why CPython's Timsort uses Insertion Sort on small/natural runs.

What Topic Is Next → Next Topic 7.15: Divide-and-Conquer Sorting (Merge Sort and Quick Sort)

7.15 Divide-and-Conquer Sorting (Merge Sort and Quick Sort) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stable `O(n log n)` Merge Sort • Randomized Quick Sort partitioning • Python Timsort `key=` functions • Interval merging • Production Edge-Case & Pitfall Guard (Divide-and-Conquer Sorting (Merge Sort and Quick Sort)) • Hands-On Python Implementation & Verification (Divide-and-Conquer Sorting (Merge Sort and Quick Sort)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Divide-and-Conquer Sorting (Merge Sort and Quick Sor...

Purpose: Merging overlapping calendar/booking intervals after sorting by start timestamp (`O(n log n)`).

What Topic Is Next → Next Topic 7.16: Recursive Problem-Solving

7.16 Recursive Problem-Solving [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Recurrence relations • Divide- • -conquer fast modular exponentiation • Recursion trees • Production Edge-Case & Pitfall Guard (Recursive Problem-Solving) • Hands-On Python Implementation & Verification (Recursive Problem-Solving) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Recursive Problem-Solving

Purpose: Evaluating recursive AST/JSON filters and hierarchical category permissions.

What Topic Is Next → Next Topic 7.17: Backtracking Search and Pruning

7.17 Backtracking Search and Pruning [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Choose -> Explore -> Unchoose (`path.append` / `dfs` / `path.pop`) template for Subsets • Permutations • Combination Sum • N-Queens • Production Edge-Case & Pitfall Guard (Backtracking Search and Pruning) • Hands-On Python Implementation & Verification (Backtracking Search and Pruning) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backtracking Search

Purpose: Generating valid shift/slot schedules and constraint-satisfying configurations with early pruning.

What Topic Is Next → Next Topic 7.18: Singly and Doubly Linked Lists

7.18 Singly and Doubly Linked Lists [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ListNode` pointer rewiring • Dummy sentinel heads • Floyd's fast/slow tortoise-hare cycle & middle detection • Doubly Linked List LRU nodes • Production Edge-Case & Pitfall Guard (Singly and Doubly Linked Lists) • Hands-On Python Implementation & Verification (Singly and Doubly Linked Lists) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Singly

Purpose: Implementing `O(1)` eviction Doubly Linked Lists inside LRU caches and connection pools.

What Topic Is Next → Next Topic 7.19: Stacks and Monotonic Stacks

7.19 Stacks and Monotonic Stacks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: LIFO stack via Python `list` (`append`/`pop`) • Balanced brackets • `MinStack` • Monotonic stack (`Next Greater Element` • Production Edge-Case & Pitfall Guard (Stacks and Monotonic Stacks) • Hands-On Python Implementation & Verification (Stacks and Monotonic Stacks) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stacks

Purpose: Validating nested expression syntax, undo stacks, and `O(n)` next-warmer/stock-span range queries.

What Topic Is Next → Next Topic 7.20: Queues and collections.deque

7.20 Queues and collections.deque [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: FIFO queue with `collections.deque` (`append` + `popleft` in `O(1)`) • Circular queues • Monotonic deque (`Sliding Window Maximum`) • Production Edge-Case & Pitfall Guard (Queues and collections.deque) • Hands-On Python Implementation & Verification (Queues and collections.deque) • Queues and collections.deque — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Queues

Purpose: Driving level-order BFS traversals and `O(n)` Sliding Window Maximum rate-spike monitors.

What Topic Is Next → Next Topic 7.21: Binary Tree Fundamentals

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

Subtopics Covered: `TreeNode(val • Left • Right)` • Height/depth • Production Edge-Case & Pitfall Guard (Binary Tree Fundamentals) • Hands-On Python Implementation & Verification (Binary Tree Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns •

Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Binary Tree Fundamentals

Purpose: Modeling expression ASTs, organizational hierarchies, and database index concepts.

What Topic Is Next → Next Topic 7.22: Binary Search Trees (BST)

7.22 Binary Search Trees (BST) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Strict BST range invariant `(min_val < node.val < max_val)` • `O(h)` search/insert/delete • Kth smallest via inorder • BST LCA • Production Edge-Case & Pitfall Guard (Binary Search Trees (BST)) • Hands-On Python Implementation & Verification (Binary Search Trees (BST)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Binary Search Trees (BST)

Purpose: Understanding ordered tree invariants underlying sorted indexes and range queries.

What Topic Is Next → Next Topic 7.23: Tree Traversals (Inorder, Preorder, Postorder, Level-Order)

7.23 Tree Traversals (Inorder, Preorder, Postorder, Level-Order) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Recursive & iterative DFS traversals (`inorder` • `preorder` • `postorder`) • `deque` BFS `levelOrder` / right-side view • Production Edge-Case & Pitfall Guard (Tree Traversals (Inorder, Preorder, Postorder, Level-Order)) • Hands-On Python Implementation & Verification (Tree Traversals (Inorder, Preorder, Postorder, Level-Order)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Tree Traversals (Inorder, Preorder, Postorder, Level...

Purpose: Serializing hierarchical category menus and evaluating bottom-up postorder aggregations.

What Topic Is Next → Next Topic 7.24: Min-Heap and Max-Heap (heapq)

7.24 Min-Heap and Max-Heap (heapq) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `heapq.heappush` • `heappop` • `heapify` (`O(n)`!) • Max-heap via `-val` negation • Production Edge-Case & Pitfall Guard (Min-Heap and Max-Heap (heapq)) • Hands-On Python Implementation & Verification (Min-Heap and Max-Heap (heapq)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Min-Heap

Purpose: Scheduling priority Celery/cron jobs, finding Top-K trending job searches in `O(n log k)`, and Dijkstra shortest paths.

What Topic Is Next → Next Topic 7.25: Graph Representations (Adjacency List and Matrix)

7.25 Graph Representations (Adjacency List and Matrix) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Directed vs undirected • Weighted vs unweighted graphs • `defaultdict(list)` adjacency lists (`O(V + E)` space) vs `V x V` matrices • Production Edge-Case & Pitfall Guard (Graph Representations (Adjacency List and Matrix)) • Hands-On Python Implementation & Verification (Graph Representations (Adjacency List and Matrix)) • Graph Representations (Adjacency List and Matrix) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Graph Representations (Adjacency List and Matrix)

Purpose: Modeling microservice call dependencies, package dependency DAGs, and city delivery networks.

What Topic Is Next → Next Topic 7.26: Breadth-First Search (BFS)

7.26 Breadth-First Search (BFS) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Shortest unweighted path via `deque` • Multi-source BFS (`Rotting Oranges`) • Kahn's in-degree Topological Sort • Production Edge-Case & Pitfall Guard (Breadth-First Search (BFS)) • Hands-On Python Implementation & Verification (Breadth-First Search (BFS)) • Breadth-First Search (BFS) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Breadth-First Search (BFS)

Purpose: Computing shortest dependency hops and detecting circular workflow dependencies via Kahn's BFS.

What Topic Is Next → Next Topic 7.27: Depth-First Search (DFS)

7.27 Depth-First Search (DFS) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Recursive • Explicit-stack DFS • Connected components • Grid flood-fill • Production Edge-Case & Pitfall Guard (Depth-First Search (DFS)) • Hands-On Python Implementation & Verification (Depth-First Search (DFS)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Depth-First Search (DFS)

Purpose: Detecting circular Alembic/service dependencies and traversing permission graphs.

What Topic Is Next → Next Topic 7.28: Greedy Algorithm Fundamentals

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

Subtopics Covered: Greedy-choice property vs DP • Earliest-finish-time interval scheduling • Jump Game reachability • Gas Station circular tour • Production Edge-Case & Pitfall Guard (Greedy Algorithm Fundamentals) • Hands-On Python Implementation & Verification (Greedy Algorithm Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Greedy Algorithm Fundamentals

Purpose: Scheduling maximum non-overlapping interview/meeting slots by sorting on earliest end time.

What Topic Is Next → Next Topic 7.29: Dynamic Programming (Memoization and Tabulation)

7.29 Dynamic Programming (Memoization and Tabulation) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Overlapping subproblems + optimal substructure • Top-down `@lru_cache` memoization vs bottom-up 1D/2D tabulation (Climbing Stairs • House Robber • Coin Change • Production Edge-Case & Pitfall Guard (Dynamic Programming (Memoization and Tabulation)) • Hands-On Python Implementation & Verification (Dynamic Programming (Memoization and Tabulation)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Programming (Memoization and Tabulation)

Purpose: Solving Coin Change, Knapsack budget allocation, and Longest Common Subsequence diffing cleanly.

What Topic Is Next → Next Topic 7.30: Bit Manipulation Problems

7.30 Bit Manipulation Problems [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: XOR self-cancellation (`Single Number`) • `n & (n - 1)` Brian Kernighan set-bit clearing • `(n & -n)` lowest set bit • Power-of-two check • Production Edge-Case & Pitfall Guard (Bit Manipulation Problems) • Hands-On Python Implementation & Verification (Bit Manipulation Problems) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Bit Manipulation Problems

Purpose: Finding unique/missing IDs in `O(1)` memory and compact bitset state masks.

What Topic Is Next → Module 7 Capstone (Week 7 Lab Build: 35-Problem Python DSA Workbench & Complexity Guide) → Next Topic 8.1: Version Control Fundamentals

Module 7 Hands-On Capstone Project (100 Marks): Week 7 Lab Build: 35-Problem Python DSA Workbench & Complexity Guide — Implement clean, type-hinted Python 3.12 solutions with documented Time/Space complexities across Arrays, Strings, Dict/Set Hashing, Two Pointers, Sliding Window, Prefix Sum, `bisect` Binary...

MODULE 8 OF 24 • STAGE 2 • 25 LIVE + LAB HOURS • 29 TOPICS (232 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Git, GitHub and Professional Development Workflow

Adopt real software-engineering team habits across every Python project: Git content-addressable architecture, config, `init`, `status`, `add`, `commit`, `log`, `diff`, `branch`, `switch`, `merge`, conflict resolution, `rebase`, `stash`, `reset` vs `revert`, remotes, `clone`/`fetch`/`pull`/`push`, GitHub Pull Requests, Python `.gitignore`, `README.md` documentation, feature-branch strategy, PR code review, Agile sprints, and Mini-Project 2.

Objectives: ✓ Navigate Working Directory, Staging Index, and `.git` Commit Graph fluently using `status`, `add`, `commit`, `log`, and `diff`. • ✓ Develop features on isolated branches (`git switch -c feat/...`), merge cleanly, resolve merge conflicts, and use `stash`, `rebase`, `reset`, and `revert` safely. • ✓ Collaborate via GitHub remotes (`clone`, `fetch`, `pull`, `push`), Pull Requests, Python `.gitignore` hygiene, and clear `README.md` setup docs. • ✓ Conduct constructive Python PR code reviews checking security, Big-O, edge cases, type hints, and logging. • ✓ Deliver Mini-Project 2: Python Inventory Management System developed across real GitHub feature branches and PRs.

8.1 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 Edge-Case & Pitfall Guard (Version Control Fundamentals) • Hands-On Python Implementation & Verification (Version Control Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 8.2: Git Architecture (Working Directory, Staging, Repository)

8.2 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 Edge-Case & Pitfall Guard (Git Architecture (Working Directory, Staging, Repository)) • Hands-On Python Implementation & Verification (Git Architecture (Working Directory, Staging, Repository)) • Git Architecture (Working Directory, Staging, Repository) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 8.3: Git Installation and Configuration

8.3 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 Edge-Case & Pitfall Guard (Git Installation and Configuration) • Hands-On Python Implementation & Verification (Git Installation and Configuration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 8.4: git init (Repository Initialization)

8.4 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 Edge-Case & Pitfall Guard (git init (Repository Initialization)) • Hands-On Python Implementation & Verification (git init (Repository Initialization)) • git init (Repository Initialization) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 8.5: git status (Inspecting Working Tree)

8.5 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 Edge-Case & Pitfall Guard (git status (Inspecting Working Tree)) • Hands-On Python Implementation & Verification (git status (Inspecting Working Tree)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 8.6: git add (Staging Changes)

8.6 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 Edge-Case & Pitfall Guard (git add (Staging Changes)) • Hands-On Python Implementation & Verification (git add (Staging Changes)) • git add (Staging Changes) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend

Developer – Industry-Ready Career Program Production Runtime 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 → Next Topic 8.7: git commit (Atomic Commits)

8.7 git commit (Atomic Commits) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Conventional Commits (`feat:` • `fix:` • `refactor:` • `test:` • Production Edge-Case & Pitfall Guard (git commit (Atomic Commits)) • Hands-On Python Implementation & Verification (git commit (Atomic Commits)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git commit (Atomic Commits)

Purpose: Enabling 1-click surgical `git revert <sha>` in production without reverting unrelated features.

What Topic Is Next → Next Topic 8.8: git log (Inspecting Commit History)

8.8 git log (Inspecting Commit History) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git log --oneline --graph --decorate --all` • `-p` patch view • `-S "symbol"` pickaxe search • File history • Production Edge-Case & Pitfall Guard (git log (Inspecting Commit History)) • Hands-On Python Implementation & Verification (git log (Inspecting Commit History)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git log (Inspecting Commit History)

Purpose: Finding the exact commit that changed tax calculation logic six weeks ago via `git log -S "tax_rate" -p`.

What Topic Is Next → Next Topic 8.9: git diff (Comparing Changes)

8.9 git diff (Comparing Changes) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unstaged `git diff` • Staged `git diff --staged` • Branch-to-branch `git diff main...feat/search` self-review • Production Edge-Case & Pitfall Guard (git diff (Comparing Changes)) • Hands-On Python Implementation & Verification (git diff (Comparing Changes)) • git diff (Comparing Changes) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git diff (Comparing Changes)

Purpose: Self-reviewing `git diff --staged` before every commit to catch stray `breakpoint()` or `print()` lines.

What Topic Is Next → Next Topic 8.10: git branch (Branch Creation)

8.10 git branch (Branch Creation) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lightweight 41-byte `HEAD` pointer mechanics • Naming conventions (`feat/...` • `fix/...` • `chore/...`) • Production Edge-Case & Pitfall Guard (git branch (Branch Creation)) • Hands-On Python Implementation & Verification (git branch (Branch Creation)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git branch (Branch Creation)

Purpose: Keeping `main` 100% deployable at all times while isolating unfinished work on feature branches.

What Topic Is Next → Next Topic 8.11: git switch (Switching Branches)

8.11 git switch (Switching Branches) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern `git switch <branch>` • `git switch -c <new-branch>` vs `git restore` (replacing ambiguous overloaded `git checkout`) • Production Edge-Case & Pitfall Guard (git switch (Switching Branches)) • Hands-On Python Implementation & Verification (git switch (Switching Branches)) • git switch (Switching Branches) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git switch (Switching Branches) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Creating and switching to a hotfix branch cleanly via `git switch -c fix/negative-stock main`.

What Topic Is Next → Next Topic 8.12: git merge (Fast-Forward and 3-Way Merges)

8.12 git merge (Fast-Forward and 3-Way Merges) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fast-forward pointer advance vs 3-way merge commit (`--no-ff`) vs Squash & Merge on Pull Requests • Production Edge-Case & Pitfall Guard (git merge (Fast-Forward and 3-Way Merges)) • Hands-On Python Implementation & Verification (git merge (Fast-Forward and 3-Way Merges)) • git merge (Fast-Forward and 3-Way Merges) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git merge (Fast-Forward and 3-Way Merges) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Integrating reviewed feature branches into `main` while running CI tests on the merged result.

What Topic Is Next → Next Topic 8.13: Merge Conflict Resolution

8.13 Merge Conflict Resolution [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading `<<<<<<< HEAD` • `=======` • `>>>>>>> branch` markers • Resolving semantic intent • Production Edge-Case & Pitfall Guard (Merge Conflict Resolution) • Hands-On Python Implementation & Verification (Merge Conflict Resolution) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Merge Conflict Resolution

Purpose: Resolving concurrent edits to `urls.py`, Alembic heads, or `pyproject.toml` without silently deleting a teammate's code.

What Topic Is Next → Next Topic 8.14: git rebase Fundamentals

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

Subtopics Covered: Replaying feature commits linearly atop latest `origin/main` (`git rebase main`) • `--continue`/`--abort` • The Golden Rule of Rebase • Production Edge-Case & Pitfall Guard (git rebase Fundamentals) • Hands-On Python Implementation & Verification (git rebase Fundamentals) • git rebase Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git rebase Fundamentals

Purpose: Keeping your open feature branch up-to-date with `main` cleanly without messy `"Merge branch main into feat"` commits.

What Topic Is Next → Next Topic 8.15: git stash (Saving Work in Progress)

8.15 git stash (Saving Work in Progress) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git stash push -u -m "wip: ..."` • `git stash list` • `git stash pop` vs `apply` • `git stash drop` • Production Edge-Case & Pitfall Guard (git stash (Saving Work in Progress)) • Hands-On Python Implementation & Verification (git stash (Saving Work in Progress)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git stash (Saving Work in Progress)

Purpose: Shelving half-finished feature edits (including untracked `-u` files) to fix an urgent production bug on `main`.

What Topic Is Next → Next Topic 8.16: git reset (Soft, Mixed and Hard)

8.16 git reset (Soft, Mixed and Hard) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `--soft` (moves `HEAD` only) • `--mixed` (unstages too) • `--hard` (DESTROYS working changes!) • `git reflog` recovery • Production Edge-Case & Pitfall Guard (git reset (Soft, Mixed and Hard)) • Hands-On Python Implementation & Verification (git reset (Soft, Mixed and Hard)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git reset (Soft, Mixed and Hard)

Purpose: Squashing the last 2 local unpushed WIP commits cleanly via `git reset --soft HEAD~2` before pushing.

What Topic Is Next → Next Topic 8.17: git revert (Safe Public Rollbacks)

8.17 git revert (Safe Public Rollbacks) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating a new inverse commit via `git revert <sha>` to undo broken commits safely on shared `main` without rewriting history • Production Edge-Case & Pitfall Guard (git revert (Safe Public Rollbacks)) • Hands-On Python Implementation & Verification (git revert (Safe Public Rollbacks)) • git revert (Safe Public Rollbacks) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git revert (Safe Public Rollbacks) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Rolling back a faulty deployment commit on `main` in 10 seconds while keeping team Git history intact.

What Topic Is Next → Next Topic 8.18: Remote Repositories Configuration

8.18 Remote Repositories Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `git remote -v` • `origin` vs `upstream` • SSH (`git@github.com:...`) vs HTTPS PAT remotes • `git remote set-url` • Production Edge-Case & Pitfall Guard (Remote Repositories Configuration) • Hands-On Python Implementation & Verification (Remote Repositories Configuration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Remote Repositories Configuration

Purpose: Linking local repositories to GitHub organization remotes over passwordless Ed25519 SSH keys.

What Topic Is Next → Next Topic 8.19: git clone

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

Subtopics Covered: Cloning full history vs shallow CI clones `git clone --depth 1` • Automatic `origin` remote setup • Post-clone `.venv` bootstrap • Production Edge-Case & Pitfall Guard (git clone) • Hands-On Python Implementation & Verification (git clone) • git clone — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git clone

Purpose: Speeding up GitHub Actions CI pipelines using shallow `--depth 1` checkouts.

What Topic Is Next → Next Topic 8.20: git fetch

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

Subtopics Covered: Safely downloading remote commits/refs (`origin/main`) via `git fetch --prune` WITHOUT mutating your local working files • Production Edge-Case & Pitfall Guard (git fetch) • Hands-On Python Implementation & Verification (git fetch) • git fetch — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git fetch • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Inspecting incoming `git log HEAD..origin/main` changes safely before merging or rebasing.

What Topic Is Next → Next Topic 8.21: git pull

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

Subtopics Covered: `git pull --ff-only` vs `git pull --rebase` • Avoiding accidental merge-bubble commits on local branches • Production Edge-Case & Pitfall Guard (git pull) • Hands-On Python Implementation & Verification (git pull) • git pull — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git pull • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Updating local `main` cleanly via `git pull --ff-only` before cutting a new feature branch.

What Topic Is Next → Next Topic 8.22: git push

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

Subtopics Covered: Publishing feature branches `git push -u origin feat/search` • Branch protection rules • `--force-with-lease` safety after local rebase • Production Edge-Case & Pitfall Guard (git push) • Hands-On Python Implementation & Verification (git push) • git push — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for git push

Purpose: Publishing feature branches to trigger automated GitHub Actions `pytest` + `ruff` checks.

What Topic Is Next → Next Topic 8.23: GitHub Repository Management

8.23 GitHub Repository Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Branch protection on `main` • Required status checks • Issue templates • Releases/tags (`v1.0.0`) • Production Edge-Case & Pitfall Guard (GitHub Repository Management) • Hands-On Python Implementation & Verification (GitHub Repository Management) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Repository Management

Purpose: Blocking any PR merge into `main` until `ruff`, `mypy`, and `pytest` CI checks turn green.

What Topic Is Next → Next Topic 8.24: Pull Requests and Review Workflow

8.24 Pull Requests and Review Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Draft vs Ready PRs • Concise PR descriptions (Summary • Changes • Test Plan) • Production Edge-Case & Pitfall Guard (Pull Requests and Review Workflow) • Hands-On Python Implementation & Verification (Pull Requests and Review Workflow) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pull Requests

Purpose: Keeping PRs focused (`< 300` lines) so senior engineers spot logic, SQL, and security bugs quickly.

What Topic Is Next → Next Topic 8.25: Python .gitignore Best Practices

8.25 Python .gitignore Best Practices [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Excluding `__pycache__/` • `*.py[cod]` • `.venv/` • `.env` • Production Edge-Case & Pitfall Guard (Python .gitignore Best Practices) • Hands-On Python Implementation & Verification (Python .gitignore Best Practices) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Python .gitignore Best Practices

Purpose: Keeping repositories lean, OS-neutral, and 100% free of credentials and virtualenvs.

What Topic Is Next → Next Topic 8.26: Technical Documentation with README.md

8.26 Technical Documentation with README.md [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Writing recruiter- • Engineer-ready `README.md`: architecture overview • Tech stack • Setup commands • Production Edge-Case & Pitfall Guard (Technical Documentation with README.md) • Hands-On Python Implementation & Verification (Technical Documentation with README.md) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Technical Documentation with README.md

Purpose: Allowing hiring managers and interviewers to understand and run your GitHub portfolio projects in under 2 minutes!

What Topic Is Next → Next Topic 8.27: Feature Branching Strategy

8.27 Feature Branching Strategy [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: GitHub Flow (short-lived feature branches -> PR -> CI -> `main`) vs GitFlow release branches • Trunk-based delivery • Production Edge-Case & Pitfall Guard (Feature Branching Strategy) • Hands-On Python Implementation & Verification (Feature Branching Strategy) • Feature Branching Strategy — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Feature Branching Strategy • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Shipping multiple daily backend deployments safely using short-lived (< 2 day) feature branches.

What Topic Is Next → Next Topic 8.28: Reviewing Python Code in PRs

8.28 Reviewing Python Code in PRs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Checking SQL injection/secrets • Mutable defaults • `N+1` queries • Missing timeouts • Production Edge-Case & Pitfall Guard (Reviewing Python Code in PRs) • Hands-On Python Implementation & Verification (Reviewing Python Code in PRs) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Reviewing Python Code in PRs

Purpose: Catching production-breaking defects during peer review before code ever reaches staging or production.

What Topic Is Next → Next Topic 8.29: Agile Fundamentals (Sprints, Backlog, Task Tracking)

8.29 Agile Fundamentals (Sprints, Backlog, Task Tracking) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: User stories • Acceptance criteria • Backlog refinement • 1–2 week Sprints • Production Edge-Case & Pitfall Guard (Agile Fundamentals (Sprints, Backlog, Task Tracking)) • Hands-On Python Implementation & Verification (Agile Fundamentals (Sprints, Backlog, Task Tracking)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Agile Fundamentals (Sprints, Backlog, Task Tracking)

Purpose: Breaking a Job Portal or Inventory epic into estimable user stories with testable acceptance criteria.

What Topic Is Next → Module 8 Capstone (Project 2: Python Inventory Management System) → Next Topic 9.1: Database and DBMS Introduction

Module 8 Hands-On Capstone Project (100 Marks): Project 2: Python Inventory Management System — Build an advanced, type-annotated, OOP Python 3.12 Inventory Management System featuring O(1) SKU indexing, multi-criteria search/filtering, low-stock priority alerts, atomic file persistenc...

MODULE 9 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 25 TOPICS (200 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Database Fundamentals and Complete SQL

Build rock-solid relational database & SQL mastery on PostgreSQL 16 (with SQLite & MySQL awareness) BEFORE touching any ORM: DBMS/RDBMS architecture, `psql` & `pgAdmin`, `CREATE DATABASE`/`TABLE`, PostgreSQL data types, `PRIMARY KEY`/`FOREIGN KEY`/`UNIQUE`/`NOT NULL`/`CHECK`/`DEFAULT` constraints, `INSERT`, `SELECT`, `WHERE`, `DISTINCT`, `ORDER BY`, `LIMIT`/`OFFSET`, `UPDATE`, `DELETE`, logical/`BETWEEN`/`LIKE` operators, `COUNT`/`SUM`/`AVG`/`MIN`/`MAX`, `GROUP BY`, `HAVING`, and aliases.

Objectives: ✓ Set up PostgreSQL 16 locally, navigate `psql` and `pgAdmin`, and explain RDBMS vs SQLite vs MySQL vs NoSQL trade-offs. • ✓ Author production `CREATE TABLE` DDL scripts using accurate numeric (`NUMERIC(12,2)`), text, `TIMESTAMPTZ`, boolean, and constraint rules. • ✓ Perform safe `INSERT ... RETURNING`, `UPDATE`, and `DELETE` DML operations (never forgetting `WHERE`!). • ✓ Write filtered, sorted, deduplicated, and paginated `SELECT` queries

handling SQL 3-valued `NULL` logic (`IS NULL`) accurately. • ✓ Build analytical summary reports using `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`, `GROUP BY`, and `HAVING`.

9.1 Database and DBMS Introduction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why flat JSON/CSV files fail under concurrent multi-user writes • Durability • Crash recovery • DBMS responsibilities • Production Edge-Case & Pitfall Guard (Database and DBMS Introduction) • Hands-On Python Implementation & Verification (Database and DBMS Introduction) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database

Purpose: Replacing Month 1–2 JSON file persistence with concurrent, crash-safe PostgreSQL ACID storage.

What Topic Is Next → Next Topic 9.2: Relational Database Architecture (RDBMS)

9.2 Relational Database Architecture (RDBMS) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Relations (tables) • Tuples (rows) • Attributes (columns) • Schemas • Production Edge-Case & Pitfall Guard (Relational Database Architecture (RDBMS)) • Hands-On Python Implementation & Verification (Relational Database Architecture (RDBMS)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Relational Database Architecture (RDBMS)

Purpose: Understanding PostgreSQL connection processes, `public` vs custom schemas, and Write-Ahead Logging (WAL).

What Topic Is Next → Next Topic 9.3: Relational vs NoSQL Databases (PostgreSQL, MySQL, SQLite)

9.3 Relational vs NoSQL Databases (PostgreSQL, MySQL, SQLite) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PostgreSQL (primary production RDBMS) • SQLite (embedded file DB for local/tests) • MySQL fundamentals • Vs document/key-value stores • Production Edge-Case & Pitfall Guard (Relational vs NoSQL Databases (PostgreSQL, MySQL, SQLite)) • Hands-On Python Implementation & Verification (Relational vs NoSQL Databases (PostgreSQL, MySQL, SQLite)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Relational vs NoSQL Databases (PostgreSQL, MySQL, SQ...

Purpose: Using PostgreSQL 16 as primary transactional DB, Redis for sub-millisecond cache/queues, and SQLite only for lightweight local prototypes.

What Topic Is Next → Next Topic 9.4: PostgreSQL Installation and Configuration

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

Subtopics Covered: Installing PostgreSQL 16 (native & Docker) • Roles/users • `postgresql.conf` • `pg_hba.conf` • Production Edge-Case & Pitfall Guard (PostgreSQL Installation and Configuration) • Hands-On Python Implementation & Verification (PostgreSQL Installation and Configuration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PostgreSQL Installation

Purpose: Creating a dedicated `app_user` role with scoped DB privileges instead of connecting web apps as superuser `postgres`!

What Topic Is Next → Next Topic 9.5: Database Tools (psql and pgAdmin)

9.5 Database Tools (psql and pgAdmin) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `psql` CLI meta-commands (`\l` • `\c` • `\dt` • `\d+ table` • Production Edge-Case & Pitfall Guard (Database Tools (psql and pgAdmin)) • Hands-On Python Implementation & Verification (Database Tools (psql and pgAdmin)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Tools (psql and pgAdmin)

Purpose: Inspecting table indexes (`\d+ employees`) and query execution times (`\timing`) over SSH on Linux servers where no GUI exists.

What Topic Is Next → Next Topic 9.6: CREATE DATABASE

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

Subtopics Covered: `CREATE DATABASE ... ENCODING 'UTF8'` • `DROP DATABASE IF EXISTS` • Isolated `dev` / `test` / `prod` databases • Production Edge-Case & Pitfall Guard (CREATE DATABASE) • Hands-On Python Implementation & Verification (CREATE DATABASE) • CREATE DATABASE — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CREATE DATABASE

Purpose: Provisioning UTF-8 encoded databases supporting international characters and emojis cleanly.

What Topic Is Next → Next Topic 9.7: CREATE TABLE

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

Subtopics Covered: `CREATE TABLE IF NOT EXISTS` • `GENERATED ALWAYS AS IDENTITY` (modern SQL standard over legacy `SERIAL`) • `ALTER TABLE` • `DROP TABLE` • Production Edge-Case & Pitfall Guard (CREATE TABLE) • Hands-On Python Implementation & Verification (CREATE TABLE) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CREATE TABLE

Purpose: Defining clean `departments` and `employees` tables using modern identity primary keys and audit timestamps.

What Topic Is Next → Next Topic 9.8: SQL Data Types (Numeric, Character, Date/Time, Boolean, UUID)

9.8 SQL Data Types (Numeric, Character, Date/Time, Boolean, UUID) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `BIGINT`/`INTEGER` • Exact `NUMERIC(12 • 2)` vs `DOUBLE PRECISION` • `VARCHAR(n)`/`TEXT` • Production Edge-Case & Pitfall Guard (SQL Data Types (Numeric, Character, Date/Time, Boolean, UUID)) • Hands-On Python Implementation & Verification (SQL Data Types (Numeric, Character, Date/Time, Boolean, UUID)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Data Types (Numeric, Character, Date/Time, Boole...

Purpose: Storing salaries/prices as `NUMERIC(12, 2)` (zero floating-point drift!) and timestamps as UTC `TIMESTAMPTZ`.

What Topic Is Next → Next Topic 9.9: Relational Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE)

9.9 Relational Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single & composite `PRIMARY KEY` • `FOREIGN KEY ... REFERENCES ... ON DELETE RESTRICT/CASCADE/SET NULL` • `UNIQUE` indexes • Production Edge-Case & Pitfall Guard (Relational Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE)) • Hands-On Python Implementation & Verification (Relational Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE)) • Relational Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Relational Constraints (PRIMARY KEY, FOREIGN KEY, UN...

Purpose: Enforcing database-level uniqueness on `email` and `(job_id, candidate_id)` so concurrent requests can NEVER create duplicate applications!

What Topic Is Next → Next Topic 9.10: Domain Constraints (NOT NULL, CHECK, DEFAULT)

9.10 Domain Constraints (NOT NULL, CHECK, DEFAULT) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `NOT NULL` mandatory columns • `CHECK (salary >= 0 • Max_salary >= min_salary)` domain invariants • Server-side `DEFAULT` values • Production Edge-Case & Pitfall Guard (Domain Constraints (NOT NULL, CHECK, DEFAULT)) • Hands-On Python Implementation & Verification (Domain Constraints (NOT NULL, CHECK, DEFAULT)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Domain Constraints (NOT NULL, CHECK, DEFAULT)

Purpose: Blocking negative inventory stock (`CHECK (stock_qty >= 0)`) at the PostgreSQL storage engine level.

What Topic Is Next → Next Topic 9.11: INSERT Statements

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

Subtopics Covered: Single-row • Multi-row batch `INSERT INTO ... VALUES` • PostgreSQL `RETURNING id` • Idempotent `ON CONFLICT DO NOTHING / DO UPDATE` upserts • Production Edge-Case & Pitfall Guard (INSERT Statements) • Hands-On Python Implementation & Verification (INSERT Statements) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for INSERT Statements

Purpose: Inserting a new employee and fetching the generated `id` + `created_at` in a single network roundtrip via `RETURNING`!

What Topic Is Next → Next Topic 9.12: SELECT Queries

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

Subtopics Covered: Explicit column projection `SELECT id • Emp_code • Email` vs `SELECT *` • Computed expressions • Production Edge-Case & Pitfall Guard (SELECT Queries) • Hands-On Python Implementation & Verification (SELECT Queries) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SELECT Queries

Purpose: Projecting only required columns over the network and handling nullable bonus fields via `salary + COALESCE(bonus, 0)`.

What Topic Is Next → Next Topic 9.13: WHERE Filtering

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

Subtopics Covered: Comparison predicates (`=` • `<>` • `<` • `<=` • Production Edge-Case & Pitfall Guard (WHERE Filtering) • Hands-On Python Implementation & Verification (WHERE Filtering) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WHERE Filtering

Purpose: Filtering active, unpublished, or soft-deleted (`WHERE deleted_at IS NULL`) records accurately.

What Topic Is Next → Next Topic 9.14: DISTINCT Deduplication

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

Subtopics Covered: `SELECT DISTINCT col1 • Col2` • PostgreSQL `SELECT DISTINCT ON (department_id) ... ORDER BY ...` latest-per-group idiom • Production Edge-Case & Pitfall Guard (DISTINCT Deduplication) • Hands-On Python Implementation & Verification (DISTINCT Deduplication) • DISTINCT Deduplication — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DISTINCT Deduplication

Purpose: Fetching distinct hiring cities for a filter dropdown or latest application status per candidate.

What Topic Is Next → Next Topic 9.15: ORDER BY Sorting

9.15 ORDER BY Sorting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-column `ASC`/`DESC` sorting • `NULLS LAST` / `NULLS FIRST` • Deterministic tie-breaker `ORDER BY created_at DESC • Id DESC` • Production Edge-Case & Pitfall Guard (ORDER BY Sorting) • Hands-On Python Implementation & Verification (ORDER BY Sorting) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ORDER BY Sorting

Purpose: Adding a unique tie-breaker `, id DESC` so paginated API pages never duplicate or skip rows with identical timestamps!

What Topic Is Next → Next Topic 9.16: LIMIT and OFFSET Pagination

9.16 LIMIT and OFFSET Pagination [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Offset pagination (`LIMIT 20 OFFSET 40`) vs Keyset/Cursor pagination (`WHERE (created_at • Id) < (...) LIMIT 20`) at deep pages • Production Edge-Case & Pitfall Guard (LIMIT and OFFSET Pagination) • Hands-On Python Implementation & Verification (LIMIT and OFFSET Pagination) • LIMIT and OFFSET Pagination — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for LIMIT • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Powering DRF & FastAPI paginated job search endpoints while understanding why `OFFSET 100000` scans and discards 100,000 rows.

What Topic Is Next → Next Topic 9.17: UPDATE Statements

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

Subtopics Covered: Targeted `UPDATE ... SET ... WHERE ... RETURNING *` • Atomic in-place increments `SET stock = stock - 1 WHERE id = $1 • Stock >= 1` • Production Edge-Case & Pitfall Guard (UPDATE Statements) • Hands-On Python Implementation & Verification (UPDATE Statements) • UPDATE Statements — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for UPDATE Statements

Purpose: Decrementing inventory stock atomically in SQL (`WHERE id = $1 AND stock >= $2`) to prevent race-condition overselling!

What Topic Is Next → Next Topic 9.18: DELETE Statements

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

Subtopics Covered: Guarded `DELETE FROM ... WHERE ... RETURNING id` • `TRUNCATE` vs `DELETE` • Soft-delete (`deleted_at TIMESTAMPTZ`) audit patterns • Production Edge-Case & Pitfall Guard (DELETE Statements) • Hands-On Python Implementation & Verification (DELETE Statements) • DELETE Statements — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DELETE Statements

Purpose: Using soft-delete (`deleted_at = NOW()`) for regulated user/order records vs hard `DELETE` for expired OTP tokens.

What Topic Is Next → Next Topic 9.19: Logical Operators (AND, OR, NOT, IN)

9.19 Logical Operators (AND, OR, NOT, IN) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: ` binds tighter than `OR` (always parenthesize `OR`!) • `IN (...)` • `NOT EXISTS` vs `NOT IN` null safety • Production Edge-Case & Pitfall Guard (Logical Operators (AND, OR, NOT, IN)) • Hands-On Python Implementation & Verification (Logical Operators (AND, OR, NOT, IN)) • Logical Operators (AND, OR, NOT, IN) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Logical Operators (AND, OR, NOT, IN)

Purpose: Combining `(is_remote = TRUE OR city = 'Bengaluru') AND is_active = TRUE` in job search filters.

What Topic Is Next → Next Topic 9.20: BETWEEN Range Filtering

9.20 BETWEEN Range Filtering [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Inclusive `BETWEEN low • High` (`col >= low • Col <= high`) vs half-open `[start • Next_day)` timestamp ranges • Production Edge-Case & Pitfall Guard (BETWEEN Range Filtering) • Hands-On Python Implementation & Verification (BETWEEN Range Filtering) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BETWEEN Range Filtering

Purpose: Querying monthly payroll/orders reliably with `created_at >= '2026-11-01' AND created_at < '2026-12-01'` so `23:59:59.999` microsecond rows are never missed!

What Topic Is Next → Next Topic 9.21: LIKE and ILIKE Pattern Matching

9.21 LIKE and ILIKE Pattern Matching [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `%` (0+ chars) • `_` (1 char) wildcards • PostgreSQL case-insensitive `ILIKE` • Escaping literal `%`/`_` • Production Edge-Case & Pitfall Guard (LIKE and ILIKE Pattern Matching) • Hands-On Python Implementation & Verification (LIKE and ILIKE Pattern Matching) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for LIKE

Purpose: Implementing case-insensitive prefix autocomplete `title ILIKE 'python%'` and understanding `pg_trgm` GIN indexes for substring search.

What Topic Is Next → Next Topic 9.22: Aggregate Functions (COUNT, SUM, AVG, MIN, MAX)

9.22 Aggregate Functions (COUNT, SUM, AVG, MIN, MAX) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `COUNT(*)` (all rows) vs `COUNT(col)` (non-null only!) vs `COUNT(DISTINCT col)` • `SUM` • `AVG` • `MIN` • Production Edge-Case & Pitfall Guard (Aggregate Functions (COUNT, SUM, AVG, MIN, MAX)) • Hands-On Python Implementation & Verification (Aggregate Functions (COUNT, SUM, AVG, MIN, MAX)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Aggregate Functions (COUNT, SUM, AVG, MIN, MAX)

Purpose: Computing total applications, shortlisted count (`COUNT(*) FILTER (WHERE status = 'shortlisted')`), and average offered CTC in one table scan!

What Topic Is Next → Next Topic 9.23: GROUP BY Aggregation

9.23 GROUP BY Aggregation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Grouping by one or multiple columns/expressions • The SQL functional-dependency rule (every non-aggregated `SELECT` column must be grouped) • Production Edge-Case & Pitfall Guard (GROUP BY Aggregation) • Hands-On Python Implementation & Verification (GROUP BY Aggregation) • GROUP BY Aggregation — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GROUP BY Aggregation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Summarizing headcount, min/max/avg salary per `department_id` and active status.

What Topic Is Next → Next Topic 9.24: HAVING Group Filtering

9.24 HAVING Group Filtering [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: SQL logical execution order (`FROM -> WHERE -> GROUP BY -> HAVING -> SELECT -> DISTINCT -> ORDER BY -> LIMIT`) • Filtering aggregated groups with `HAVING` • Production Edge-Case & Pitfall Guard (HAVING Group Filtering) • Hands-On Python Implementation & Verification (HAVING Group Filtering) • HAVING Group Filtering — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HAVING Group Filtering • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Finding departments with `COUNT(*) >= 5` active engineers or duplicate emails `HAVING COUNT(*) > 1`.

What Topic Is Next → Next Topic 9.25: Column and Table Aliases

9.25 Column and Table Aliases [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Readable column aliases (`AS monthly_ctc`) • Concise table aliases (`employees AS e` • `departments AS d`) across joins • Subqueries • Production Edge-Case & Pitfall Guard (Column and Table Aliases) • Hands-On Python Implementation & Verification (Column and Table Aliases) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Column

Purpose: Mapping SQL output columns cleanly to Python dictionary/DTO field names and disambiguating joined `e.id` vs `d.id` columns.

What Topic Is Next → Module 9 Capstone (Week 9 Lab Build: PostgreSQL HR & Recruitment DDL + 35-Query Pack) → Next Topic 10.1: INNER JOIN

Module 9 Hands-On Capstone Project (100 Marks): Week 9 Lab Build: PostgreSQL HR & Recruitment DDL + 35-Query Pack — Design constrained PostgreSQL 16 tables (`departments`, `employees`, `job_postings`, `job_applications`) with `IDENTITY` PKs, FKs, `UNIQUE`, `CHECK` constraints, and 35 analytical queries (`...

MODULE 10 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 30 TOPICS (240 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Advanced SQL and Database Design

Master interview- and production-grade PostgreSQL engineering: `INNER`/`LEFT`/`RIGHT`/`FULL OUTER`/`CROSS`/`SELF` joins, single/multi-row & correlated subqueries, CTEs, recursive CTEs, `OVER`/`PARTITION BY` window functions (`ROW_NUMBER`, `RANK`, `DENSE_RANK`, `LAG`, `LEAD`, running totals), views, materialized views, B-Tree & composite indexes, `EXPLAIN ANALYZE`, `COMMIT`/`ROLLBACK`, ACID, isolation levels, row/table locking, deadlocks, 1NF–3NF normalization, 1:1/1:N/M:N ER diagrams, referential integrity, stored procedures, triggers, and `pg_dump` backups.

Objectives: ✓ Combine multi-table datasets accurately using `INNER`, `LEFT`, `FULL OUTER`, and `SELF` joins without unintended row multiplication or `LEFT JOIN` `WHERE` traps. • ✓ Solve Top-N per department, running totals, month-over-month deltas, and org-chart hierarchies using CTEs, Recursive CTEs, and Window Functions. • ✓ Design 3NF normalized ER schemas (`1:1`, `1:N`, `M:N` junction tables) and tune slow queries with composite B-Tree indexes and `EXPLAIN ANALYZE`. • ✓ Guarantee data consistency under concurrency using ACID transactions, isolation levels, `SELECT ... FOR UPDATE` row locks, and deadlock prevention. • ✓ Understand views, materialized views, stored procedures, triggers, and PostgreSQL `pg_dump`/`pg_restore` backup operations.

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

Subtopics Covered: Equi-joins across 2+ tables (`JOIN ... ON e.department_id = d.id`) returning rows with matching keys in both tables • Production Edge-Case & Pitfall Guard (INNER JOIN) • Hands-On Python Implementation & Verification (INNER JOIN) • INNER JOIN — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for INNER JOIN • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Fetching active employees together with their assigned department code and name.

What Topic Is Next → Next Topic 10.2: LEFT JOIN

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

Subtopics Covered: Preserving all left-table rows (`NULL`-extending unmatched right columns) • Anti-join `WHERE r.id IS NULL` • `ON` vs `WHERE` filter placement • Production Edge-Case & Pitfall Guard (LEFT JOIN) • Hands-On Python Implementation & Verification (LEFT JOIN) • LEFT JOIN — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for LEFT JOIN

Purpose: Listing ALL departments including newly created departments with `0` employees (`COUNT(e.id)` — NOT `COUNT(*)`!).

What Topic Is Next → Next Topic 10.3: RIGHT JOIN

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

Subtopics Covered: `RIGHT OUTER JOIN` mechanics • Why rewriting `A RIGHT JOIN B` as `B LEFT JOIN A` improves query readability • Production Edge-Case & Pitfall Guard (RIGHT JOIN) • Hands-On Python Implementation & Verification (RIGHT JOIN) • RIGHT JOIN — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for RIGHT JOIN • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Reading legacy reporting SQL containing `RIGHT JOIN` and refactoring to consistent top-down `LEFT JOIN` style.

What Topic Is Next → Next Topic 10.4: FULL OUTER JOIN

10.4 FULL OUTER JOIN [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Preserving unmatched rows from BOTH sides (`FULL OUTER JOIN`) with `COALESCE(a.key • B.key)` for financial reconciliation • Production Edge-Case & Pitfall Guard (FULL OUTER JOIN) • Hands-On Python Implementation & Verification (FULL OUTER JOIN) • FULL OUTER JOIN — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for FULL OUTER JOIN • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Reconciling internal `orders` ledger rows against payment-gateway `settlements` to spot missing rows on either side.

What Topic Is Next → Next Topic 10.5: CROSS JOIN

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

Subtopics Covered: Intentional `M x N` Cartesian product (`CROSS JOIN` + `generate_series`) for date-spine matrix reports vs accidental runaway joins • Production Edge-Case & Pitfall Guard (CROSS JOIN) • Hands-On Python Implementation & Verification (CROSS JOIN) • CROSS JOIN — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CROSS JOIN • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Generating a complete `(department x calendar_month)` grid via `CROSS JOIN` so zero-hire months appear as `0` instead of missing!

What Topic Is Next → Next Topic 10.6: SELF JOIN

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

Subtopics Covered: Joining a table to itself using mandatory distinct aliases (`employees AS emp LEFT JOIN employees AS mgr ON mgr.id = emp.manager_id`) • Production Edge-Case & Pitfall Guard (SELF JOIN) • Hands-On Python Implementation & Verification (SELF JOIN) • SELF JOIN — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SELF JOIN • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Listing every employee alongside their direct reporting manager's name and finding employees earning more than their manager.

What Topic Is Next → Next Topic 10.7: Single-Row and Multi-Row Subqueries

10.7 Single-Row and Multi-Row Subqueries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Scalar subqueries (`= (SELECT AVG(salary) ...)`) • Multi-row `IN` / `ANY` / `ALL` • Derived inline tables in `FROM` • Production Edge-Case & Pitfall Guard (Single-Row and Multi-Row Subqueries) • Hands-On Python Implementation & Verification (Single-Row and Multi-Row Subqueries) • Single-Row and Multi-Row Subqueries — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Single-Row

Purpose: Finding all employees earning above company-wide average salary.

What Topic Is Next → Next Topic 10.8: Correlated Subqueries

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

Subtopics Covered: Inner subqueries referencing outer-row columns (`EXISTS (SELECT 1 FROM job_applications a WHERE a.job_id = j.id)`) • Semi-join optimization • Production Edge-Case & Pitfall Guard (Correlated Subqueries) • Hands-On Python Implementation & Verification (Correlated Subqueries) • Correlated Subqueries — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Correlated Subqueries • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Filtering employers who have at least one published job via `WHERE EXISTS (SELECT 1 ...)` without row duplication!

What Topic Is Next → Next Topic 10.9: Common Table Expressions (CTE)

10.9 Common Table Expressions (CTE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `WITH cte_name AS (...)` modular named sub-queries for readable multi-stage analytical pipelines • Data-modifying CTEs • Production Edge-Case & Pitfall Guard (Common Table Expressions (CTE)) • Hands-On Python Implementation & Verification (Common Table Expressions (CTE)) • Common Table Expressions (CTE) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Common Table Expressions (CTE) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Breaking a 60-line hiring-funnel conversion report into 3 self-documenting `WITH` stages (`job_stats`, `shortlisted`, `final_rates`).

What Topic Is Next → Next Topic 10.10: Recursive CTEs for Hierarchical Data

10.10 Recursive CTEs for Hierarchical Data [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `WITH RECURSIVE` anchor member + `UNION ALL` recursive member + depth guard for org charts • Nested category trees • Production Edge-Case & Pitfall Guard (Recursive CTEs for Hierarchical Data) • Hands-On Python Implementation & Verification (Recursive CTEs for Hierarchical Data) • Recursive CTEs for Hierarchical Data — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Recursive CTEs for Hierarchical Data • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Fetching an entire reporting hierarchy beneath a VP or all nested sub-categories in a single SQL query!

What Topic Is Next → Next Topic 10.11: Window Functions (OVER and PARTITION BY)

10.11 Window Functions (OVER and PARTITION BY) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Computing per-partition metrics via `AVG(salary) OVER (PARTITION BY department_id)` WITHOUT collapsing detail rows like `GROUP BY` • Production Edge-Case & Pitfall Guard (Window Functions (OVER and PARTITION BY)) • Hands-On Python Implementation & Verification (Window Functions (OVER and PARTITION BY)) • Window Functions (OVER and PARTITION BY) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Window Functions (OVER and PARTITION BY) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Showing each employee's individual salary right beside their department average and % share of department payroll in one pass.

What Topic Is Next → Next Topic 10.12: Ranking Window Functions (ROW_NUMBER, RANK, DENSE_RANK)

10.12 Ranking Window Functions (ROW_NUMBER, RANK, DENSE_RANK) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exact tie differences: `ROW_NUMBER()` (`1 • 4`) • `RANK()` (`1 • 4` gap) • Production Edge-Case & Pitfall Guard (Ranking Window Functions (ROW_NUMBER, RANK, DENSE_RANK)) • Hands-On Python Implementation & Verification (Ranking Window Functions (ROW_NUMBER, RANK, DENSE_RANK)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Ranking Window Functions (ROW_NUMBER, RANK, DENSE_RA...

Purpose: Solving the #1 SQL interview question: "Find the 2nd / Top-3 highest distinct salaries in EACH department" (`DENSE_RANK() <= 3`)!

What Topic Is Next → Next Topic 10.13: Analytical Window Functions (LAG, LEAD, Running Totals)

10.13 Analytical Window Functions (LAG, LEAD, Running Totals) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `LAG(col • 0) OVER (...)` • `LEAD()` • Cumulative running totals `SUM(amt) OVER (ORDER BY dt ROWS BETWEEN UNBOUNDED PRECEDING • Production Edge-Case & Pitfall Guard (Analytical Window Functions (LAG, LEAD, Running Totals)) • Hands-On Python Implementation & Verification (Analytical Window Functions (LAG, LEAD, Running Totals)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Analytical Window Functions (LAG, LEAD, Running Tota...

Purpose: Computing month-over-month job application growth and cumulative running revenue ledgers.

What Topic Is Next → Next Topic 10.14: Database Views

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

Subtopics Covered: `CREATE OR REPLACE VIEW` stored virtual queries for reusable joins • Column masking • Stable reporting contracts • Production Edge-Case & Pitfall Guard (Database Views) • Hands-On Python Implementation & Verification (Database Views) • Database Views — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Views

Purpose: Exposing `v_public_employees` omitting sensitive salary/PAN columns for reporting read roles.

What Topic Is Next → Next Topic 10.15: Materialized Views

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

Subtopics Covered: `CREATE MATERIALIZED VIEW` • Physical precomputed storage • Indexing materialized views • `REFRESH MATERIALIZED VIEW CONCURRENTLY` • Production Edge-Case & Pitfall Guard (Materialized Views) • Hands-On Python Implementation & Verification (Materialized Views) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Materialized Views

Purpose: Serving sub-5ms admin analytics dashboards backed by a materialized view refreshed every 5 minutes via Celery Beat.

What Topic Is Next → Next Topic 10.16: B-Tree Indexes and Query Performance

10.16 B-Tree Indexes and Query Performance [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: B-Tree balanced tree `O(log N)` lookups vs `O(N)` `Seq Scan` • Indexing foreign keys • Partial indexes (`WHERE is_active = TRUE`) • Write overhead • Production Edge-Case & Pitfall Guard (B-Tree Indexes and Query Performance) • Hands-On Python Implementation & Verification (B-Tree Indexes and Query Performance) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for B-Tree Indexes

Purpose: ALWAYS adding explicit B-Tree indexes on `FOREIGN KEY` columns (`PostgreSQL` auto-indexes PK/UNIQUE, NOT foreign keys!).

What Topic Is Next → Next Topic 10.17: Composite Multi-Column Indexes

10.17 Composite Multi-Column Indexes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Leftmost-prefix rule `(a • C)` • Equality columns BEFORE range/sort columns (`(status • City • Production Edge-Case & Pitfall Guard (Composite Multi-Column Indexes) • Hands-On Python Implementation & Verification (Composite Multi-Column Indexes) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Composite Multi-Column Indexes

Purpose: Designing `idx_jobs_status_city_created (status, city, created_at DESC)` so filtered + sorted job feeds hit zero heap sort!

What Topic Is Next → Next Topic 10.18: Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)

10.18 Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Reading `EXPLAIN (ANALYZE • BUFFERS)` bottom-up: `Seq Scan` • `Index Scan` • `Index Only Scan` • Production Edge-Case & Pitfall Guard (Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)) • Hands-On Python Implementation & Verification (Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Execution Plans (EXPLAIN and EXPLAIN ANALYZE)

Purpose: Cutting a 620ms sequential-scan job search down to 1.8ms `Index Scan` verified with `EXPLAIN (ANALYZE, BUFFERS)`.

What Topic Is Next → Next Topic 10.19: SQL Transactions (BEGIN, COMMIT, ROLLBACK)

10.19 SQL Transactions (BEGIN, COMMIT, ROLLBACK) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: All-or-nothing multi-statement units (`BEGIN` • `COMMIT` • `ROLLBACK` • `SAVEPOINT`) • Production Edge-Case & Pitfall Guard (SQL Transactions (BEGIN, COMMIT, ROLLBACK)) • Hands-On Python Implementation & Verification (SQL Transactions (BEGIN, COMMIT, ROLLBACK)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Transactions (BEGIN, COMMIT, ROLLBACK)

Purpose: Deducting buyer wallet, crediting seller ledger, and creating order record atomically so partial failures roll back completely.

What Topic Is Next → Next Topic 10.20: ACID Properties

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

Subtopics Covered: Atomicity (undo/rollback) • Consistency (constraints/invariants) • Isolation (MVCC snapshots) • Durability (WAL fsync on disk) • Production Edge-Case & Pitfall Guard (ACID Properties) • Hands-On Python Implementation & Verification (ACID Properties) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ACID Properties

Purpose: Explaining in backend interviews how PostgreSQL MVCC (Multi-Version Concurrency Control) lets readers never block writers.

What Topic Is Next → Next Topic 10.21: Transaction Isolation Levels

10.21 Transaction Isolation Levels [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dirty Read • Non-Repeatable Read • Phantom Read • Write Skew anomalies across `READ COMMITTED` (PostgreSQL default) • Production Edge-Case & Pitfall Guard (Transaction Isolation Levels) • Hands-On Python Implementation & Verification (Transaction Isolation Levels) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transaction Isolation Levels

Purpose: Using `REPEATABLE READ` for consistent multi-query financial reconciliation snapshots.

What Topic Is Next → Next Topic 10.22: Row-Level and Table-Level Locking

10.22 Row-Level and Table-Level Locking [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pessimistic `SELECT ... FOR UPDATE` / `FOR UPDATE SKIP LOCKED` row locks vs optimistic `version` counter checks (`WHERE id = $1 • Version = $2`) • Production Edge-Case & Pitfall Guard (Row-Level and Table-Level Locking) • Hands-On Python Implementation & Verification (Row-Level and Table-Level Locking) • Row-Level and Table-Level Locking — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Row-Level • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Using `FOR UPDATE` when booking the last seat/coupon and `FOR UPDATE SKIP LOCKED` for DB-backed job queues!

What Topic Is Next → Next Topic 10.23: Database Deadlock Fundamentals

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

Subtopics Covered: Why two concurrent transactions updating rows `(1 • 2)` vs `(2 • 1)` deadlock • PostgreSQL deadlock detector (`deadlock_timeout`) • Production Edge-Case & Pitfall Guard (Database Deadlock Fundamentals) • Hands-On Python Implementation & Verification (Database Deadlock Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program

Production Runtime Execution & Memory Lifecycle for Database Deadlock Fundamentals

Purpose: Sorting bulk-updated row IDs ascending (`ORDER BY id`) before `FOR UPDATE` / `UPDATE` so deadlocks cannot occur.

What Topic Is Next → Next Topic 10.24: Database Normalization (1NF, 2NF, 3NF)

10.24 Database Normalization (1NF, 2NF, 3NF) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 1NF (atomic columns • No comma-separated `"Python • SQL"` lists!) • 2NF (no partial composite-key dependency) • Production Edge-Case & Pitfall Guard (Database Normalization (1NF, 2NF, 3NF)) • Hands-On Python Implementation & Verification (Database Normalization (1NF, 2NF, 3NF)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Normalization (1NF, 2NF, 3NF)

Purpose: Refactoring a messy spreadsheet table (`emp_id, dept_name, dept_floor, skills_csv`) into clean 3NF relational tables.

What Topic Is Next → Next Topic 10.25: Entity Relationships (One-to-One, One-to-Many, Many-to-Many)

10.25 Entity Relationships (One-to-One, One-to-Many, Many-to-Many) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 1:1 (`UNIQUE` FK on `candidate_profiles.user_id`) • 1:N (`employees.department_id`) • M:N junction tables (`candidate_skills(candidate_id • Skill_id)` composite PK) • Production Edge-Case & Pitfall Guard (Entity Relationships (One-to-One, One-to-Many, Many-to-Many)) • Hands-On Python Implementation & Verification (Entity Relationships (One-to-One, One-to-Many, Many-to-Many)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Entity Relationships (One-to-One, One-to-Many, Many-...

Purpose: Modeling `users <-> candidate_profiles` (1:1), `employers -> jobs` (1:N), and `jobs <-> skills` (M:N) cleanly.

What Topic Is Next → Next Topic 10.26: ER Diagrams and Schema Design

10.26 ER Diagrams and Schema Design [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Crow’s Foot cardinality notation (`||--o{` • `}o--o{`) • Associative entities carrying metadata (`applied_at` • `status`) • Production Edge-Case & Pitfall Guard (ER Diagrams and Schema Design) • Hands-On Python Implementation & Verification (ER Diagrams and Schema Design) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ER Diagrams

Purpose: Creating clear ER diagrams reviewed before writing SQLAlchemy/Django migrations.

What Topic Is Next → Next Topic 10.27: Referential Integrity and Cascading Rules

10.27 Referential Integrity and Cascading Rules [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Choosing deliberately between `ON DELETE RESTRICT` (financial/orders) • `ON DELETE CASCADE` (owned child items) • `ON DELETE SET NULL` (optional manager) • Production Edge-Case & Pitfall Guard (Referential Integrity and Cascading Rules) • Hands-On Python Implementation & Verification (Referential Integrity and Cascading Rules) • Referential Integrity and Cascading Rules — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Referential Integrity

Purpose: Using `ON DELETE RESTRICT` on `invoices.customer_id` so deleting a customer can NEVER silently wipe financial invoices!

What Topic Is Next → Next Topic 10.28: Stored Procedures and Functions Fundamentals

10.28 Stored Procedures and Functions Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PL/pgSQL `CREATE FUNCTION` vs `CREATE PROCEDURE` • When DB routines help vs why business logic primarily lives in versioned Python services • Production Edge-Case & Pitfall Guard (Stored Procedures and Functions Fundamentals) • Hands-On Python Implementation & Verification (Stored Procedures and Functions Fundamentals) • Stored Procedures and Functions Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stored Procedures • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Writing lightweight `updated_at` timestamp helper functions while keeping core domain workflows testable in Python.

What Topic Is Next → Next Topic 10.29: Database Triggers Fundamentals

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

Subtopics Covered: `BEFORE UPDATE` / `AFTER INSERT` row-level triggers (`NEW` • `OLD`) for automatic `updated_at` stamps • Immutable audit trails • Production Edge-Case & Pitfall Guard (Database Triggers Fundamentals) • Hands-On Python Implementation & Verification (Database Triggers Fundamentals) • Database Triggers Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Triggers Fundamentals

Purpose: Auto-updating `updated_at = NOW()` on every row modification regardless of whether Django, FastAPI, or `psql` ran the query.

What Topic Is Next → Next Topic 10.30: PostgreSQL Backup and Restore (pg_dump and pg_restore)

10.30 PostgreSQL Backup and Restore (pg_dump and pg_restore) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Logical backups via `pg_dump -Fc` custom compressed format • `pg_restore` • Point-in-time WAL concepts • Verifying restores regularly • Production Edge-Case & Pitfall Guard (PostgreSQL Backup and Restore (pg_dump and pg_restore)) • Hands-On Python Implementation & Verification (PostgreSQL Backup and Restore (pg_dump and pg_restore)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PostgreSQL Backup

Purpose: Taking pre-migration production snapshots (`pg_dump -Fc`) and testing full recovery into a staging DB.

What Topic Is Next → Module 10 Capstone (Week 10 Lab Build: 3NF ER Schema, EXPLAIN ANALYZE Index Tuning & Window SQL) → Next Topic 11.1: Python Database Drivers (DB-API 2.0)

Module 10 Hands-On Capstone Project (100 Marks): Week 10 Lab Build: 3NF ER Schema, EXPLAIN ANALYZE Index Tuning & Window SQL — Normalize an unnormalized HR/Job dataset into 3NF, draw Crow’s Foot ER diagrams, prove composite B-Tree index speedups on 50k rows with `EXPLAIN (ANALYZE, BUFFERS)`, and author `DENSE_RANK`,...

MODULE 11 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 23 TOPICS (184 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 11: Python Database Connectivity and SQLAlchemy

Bridge Python 3.12 and PostgreSQL cleanly from low-level DB-API 2.0 drivers to modern SQLAlchemy 2.0 ORM & Alembic migrations: `psycopg` 3 connections, cursors, CRUD, parameterized SQL injection prevention, transactions, connection pooling, context managers, SQLAlchemy Core vs ORM 2.0, `Engine`, `Session` lifecycle, `DeclarativeBase` & `Mapped` typed models, relationships, `select()` queries, lazy vs eager loading (`joinedload`/`selectinload`), fixing `N+1` queries, `Alembic` migrations, and DB testing.

Objectives: ✓ Connect Python directly to PostgreSQL via `psycopg` 3 using context managers, connection pools, and 100% parameterized queries. • ✓ Understand SQLAlchemy 2.0 architecture distinguishing Core Expression Language from typed `Mapped[...]` Declarative ORM models. • ✓ Manage `Engine` pools and Unit-of-Work `Session` lifecycles cleanly without leaking connections or detached-instance bugs. • ✓ Detect and eliminate ORM `N+1` query storms using `joinedload()` (to-one) and `selectinload()` (to-many) eager loading. • ✓ Version-control database schema evolution safely with `Alembic` (`revision --autogenerate`, `upgrade head`, `downgrade -1`) and isolated DB tests.

11.1 Python Database Drivers (DB-API 2.0) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: PEP 249 DB-API 2.0 standard (`connect` • `cursor` • `execute` • `commit` • Production Edge-Case & Pitfall Guard (Python Database Drivers (DB-API 2.0)) • Hands-On Python Implementation & Verification (Python Database Drivers (DB-API 2.0)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Python Database Drivers (DB-API 2.0)

Purpose: Understanding the common Python driver contract sitting underneath SQLAlchemy and Django ORM.

What Topic Is Next → Next Topic 11.2: PostgreSQL Connectivity with psycopg 3

11.2 PostgreSQL Connectivity with psycopg 3 [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `psycopg.connect(conninfo)` • `dict_row` row factories returning typed dicts instead of positional tuples • Environment DSN loading • Production Edge-Case & Pitfall Guard (PostgreSQL Connectivity with psycopg 3) • Hands-On Python Implementation & Verification (PostgreSQL Connectivity with psycopg 3) • PostgreSQL Connectivity with psycopg 3 — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for PostgreSQL Connectivity with psycopg 3

Purpose: Connecting high-speed batch scripts and health-check probes directly to PostgreSQL via `DATABASE_URL`.

What Topic Is Next → Next Topic 11.3: Database Connection Lifecycle Management

11.3 Database Connection Lifecycle Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: TCP + TLS + PostgreSQL backend process cost per connection • `connect_timeout` • Idle-in-transaction timeouts • Clean `close()` • Production Edge-Case & Pitfall Guard (Database Connection Lifecycle Management) • Hands-On Python Implementation & Verification (Database Connection Lifecycle Management) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Connection Lifecycle Management

Purpose: Preventing `FATAL: sorry, too many clients already` errors by bounding connection lifetimes and closing cleanly.

What Topic Is Next → Next Topic 11.4: Cursors and Executing SQL Statements

11.4 Cursors and Executing SQL Statements [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `with conn.cursor() as cur:` • `cur.execute()` • `executemany()` • `fetchone()` • Production Edge-Case & Pitfall Guard (Cursors and Executing SQL Statements) • Hands-On Python Implementation & Verification (Cursors and Executing SQL Statements) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cursors

Purpose: Iterating large result sets in bounded batches via `fetchmany(500)` instead of `.fetchall()` OOM spikes.

What Topic Is Next → Next Topic 11.5: CRUD Operations Using Raw Python

11.5 CRUD Operations Using Raw Python [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing clean repository `create` • `get_by_id` • `list_paginated` • `update` • Production Edge-Case & Pitfall Guard (CRUD Operations Using Raw Python) • Hands-On Python Implementation & Verification (CRUD Operations Using Raw Python) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CRUD Operations Using Raw Python

Purpose: Understanding exact SQL + `RETURNING` mechanics that ORMs generate under the hood.

What Topic Is Next → Next Topic 11.6: Parameterized Queries and SQL Injection Prevention

11.6 Parameterized Queries and SQL Injection Prevention [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Driver parameter binding (`%s` / `%(name)s` in `psycopg` • `?` in `sqlite3`) keeping SQL AST separate from untrusted user data • Production Edge-Case & Pitfall Guard (Parameterized Queries and SQL Injection Prevention) • Hands-On Python Implementation & Verification (Parameterized Queries and SQL Injection Prevention) • Parameterized Queries and SQL Injection Prevention — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Parameterized Queries • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Blocking `' OR '1'='1'; DROP TABLE users; --` SQL Injection attacks 100% by passing parameters as the second argument to `execute(sql, params)`!

What Topic Is Next → Next Topic 11.7: Explicit Transaction Commit and Rollback

11.7 Explicit Transaction Commit and Rollback [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Grouping multi-table mutations inside `with conn.transaction():` so exceptions automatically `ROLLBACK` all partial writes • Production Edge-Case & Pitfall Guard (Explicit Transaction Commit and Rollback) • Hands-On Python Implementation & Verification (Explicit Transaction Commit and Rollback) • Explicit Transaction Commit and Rollback — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Explicit Transaction Commit • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry,

profiler traces & CI checks

Purpose: Ensuring employee creation + initial salary ledger entry either both commit or both roll back together.

What Topic Is Next → Next Topic 11.8: Database Connection Pooling

11.8 Database Connection Pooling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `psycopg_pool.ConnectionPool` • SQLAlchemy `QueuePool` (`pool_size` • `max_overflow` • `pool_pre_ping=True` • Production Edge-Case & Pitfall Guard (Database Connection Pooling) • Hands-On Python Implementation & Verification (Database Connection Pooling) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Connection Pooling

Purpose: Eliminating 25ms TCP+TLS handshake overhead per query and auto-healing stale firewall-dropped sockets via `pool_pre_ping=True`.

What Topic Is Next → Next Topic 11.9: Context Managers for Database Resources

11.9 Context Managers for Database Resources [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Nested `with pool.connection() as conn:` • `with Session(engine) as session:` guaranteeing connection return to pool on exit • Production Edge-Case & Pitfall Guard (Context Managers for Database Resources) • Hands-On Python Implementation & Verification (Context Managers for Database Resources) • Context Managers for Database Resources — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Context Managers for Database Resources • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Ensuring an unhandled `ValueError` in business code never leaks a checked-out connection from the pool.

What Topic Is Next → Next Topic 11.10: SQLAlchemy 2.0 Introduction and Architecture

11.10 SQLAlchemy 2.0 Introduction and Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unified SQLAlchemy 2.0 architecture: Dialect + Pool -> Core SQL Expression Language -> ORM Unit of Work & Identity Map • Production Edge-Case & Pitfall Guard (SQLAlchemy 2.0 Introduction and Architecture) • Hands-On Python Implementation & Verification (SQLAlchemy 2.0 Introduction and Architecture) • SQLAlchemy 2.0 Introduction and Architecture — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQLAlchemy 2.0 Introduction • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Using one industry-standard Python toolkit supporting both high-level ORM entities and raw analytical Core queries.

What Topic Is Next → Next Topic 11.11: SQLAlchemy Core (SQL Expression Language)

11.11 SQLAlchemy Core (SQL Expression Language) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `MetaData` • `Table` • Composable Pythonic `select()` • `insert()` • Production Edge-Case & Pitfall Guard (SQLAlchemy Core (SQL Expression Language)) • Hands-On Python Implementation & Verification (SQLAlchemy Core (SQL Expression Language)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQLAlchemy Core (SQL Expression Language)

Purpose: Running high-throughput bulk analytical aggregations and dynamic multi-filter report queries cleanly.

What Topic Is Next → Next Topic 11.12: SQLAlchemy ORM Concepts

11.12 SQLAlchemy ORM Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Data Mapper pattern (unlike Active Record) • Identity Map (`1` Python instance per PK per `Session`) • Dirty tracking • `flush()` vs `commit()` • Production Edge-Case & Pitfall Guard (SQLAlchemy ORM Concepts) • Hands-On Python Implementation & Verification (SQLAlchemy ORM Concepts) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQLAlchemy ORM Concepts

Purpose: Understanding why `flush()` sends SQL `INSERT` within the open transaction (populating `emp.id`) while `commit()` finalizes durability.

What Topic Is Next → Next Topic 11.13: Database Engine Configuration (create_engine)

11.13 Database Engine Configuration (create_engine) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `create_engine(DATABASE_URL • Pool_pre_ping=True • Pool_size=10 • Echo=False)` — creating ONE module-singleton `Engine` per process • Production Edge-Case & Pitfall Guard (Database Engine Configuration (create_engine)) • Hands-On Python Implementation & Verification (Database Engine Configuration (create_engine)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Engine Configuration (create_engine)

Purpose: Configuring `pool_pre_ping=True` in production and toggling `echo=True` locally to inspect generated SQL.

What Topic Is Next → Next Topic 11.14: ORM Session Lifecycle (Session and sessionmaker)

11.14 ORM Session Lifecycle (Session and sessionmaker) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SessionLocal = sessionmaker(bind=engine • Autoflush=False • Expire_on_commit=False)` • One `Session` per request/task (NEVER shared across threads!) • Production Edge-Case & Pitfall Guard (ORM Session Lifecycle (Session and sessionmaker)) • Hands-On Python Implementation & Verification (ORM Session Lifecycle (Session and sessionmaker)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ORM Session Lifecycle (Session and sessionmaker)

Purpose: Setting `expire_on_commit=False` in async/API services so reading `emp.id` after `commit()` doesn't trigger implicit lazy queries.

What Topic Is Next → Next Topic 11.15: Declarative Models (DeclarativeBase and Mapped)

11.15 Declarative Models (DeclarativeBase and Mapped) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `class Base(DeclarativeBase): pass` • `__tablename__` • `Mapped[int] = mapped_column(primary_key=True)` • `Mapped[str | None]` • Production Edge-Case & Pitfall Guard (Declarative Models (DeclarativeBase and Mapped)) • Hands-On Python Implementation & Verification (Declarative Models (DeclarativeBase and Mapped)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Declarative Models (DeclarativeBase and Mapped)

Purpose: Getting 100% static `mypy` / VS Code Pylance type inference on `Employee.email` and `Department.name`!

What Topic Is Next → Next Topic 11.16: ORM Relationships (relationship and ForeignKey)

11.16 ORM Relationships (relationship and ForeignKey) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ForeignKey("departments.id")` • Bidirectional `relationship(back_populates=...)` • `1:1` (`uselist=False`) • `1:N` • Production Edge-Case & Pitfall Guard (ORM Relationships (relationship and ForeignKey)) • Hands-On Python Implementation & Verification (ORM Relationships (relationship and ForeignKey)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ORM Relationships (relationship and ForeignKey)

Purpose: Navigating `dept.employees` and `emp.department` cleanly with explicit `back_populates` synchronization.

What Topic Is Next → Next Topic 11.17: Querying with select(), where() and scalars()

11.17 Querying with select(), where() and scalars() [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `session.scalars(select(Employee).where(...).order_by(...)).all()` • `.scalar_one_or_none()` • `session.get(Employee • Pk)` • Production Edge-Case & Pitfall Guard (Querying with select(), where() and scalars()) • Hands-On Python Implementation & Verification (Querying with select(), where() and scalars()) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Querying with select()

Purpose: Writing clean, composable 2.0 queries for search, filtering, sorting, and pagination.

What Topic Is Next → Next Topic 11.18: Lazy Loading Mechanics

11.18 Lazy Loading Mechanics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How default `lazy="select"` fires an extra SQL query on attribute access `emp.department.name` • `DetachedInstanceError` outside session scope (plus `MissingGreenlet` in async!) • Production Edge-Case & Pitfall Guard (Lazy Loading Mechanics) • Hands-On Python Implementation & Verification (Lazy Loading Mechanics) • Lazy Loading Mechanics — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Lazy Loading Mechanics • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Understanding why accessing unloaded relationships after `session.close()` or inside `AsyncSession` raises errors unless eager-loaded!

What Topic Is Next → Next Topic 11.19: Eager Loading (joinedload and selectinload)

11.19 Eager Loading (joinedload and selectinload) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `options(joinedload(Employee.department))` (`LEFT OUTER JOIN` for many-to-one/1:1) vs `options(selectinload(Department.employees))` (`WHERE id IN (...)` for 1:N/M:N) • Production Edge-Case & Pitfall Guard (Eager Loading (joinedload and selectinload)) • Hands-On Python Implementation & Verification (Eager Loading (joinedload and selectinload)) • Eager Loading (joinedload and selectinload) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Eager Loading (joinedload and selectinload) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Loading 50 departments and all 400 of their employees in exactly TWO SQL queries via `selectinload`!

What Topic Is Next → Next Topic 11.20: Identifying and Fixing the N+1 Query Problem

11.20 Identifying and Fixing the N+1 Query Problem [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spotting `1` list query + `N` per-row loop queries (`101` SQL roundtrips for `100` items!) via SQL logging/event counters • Fixing down to `1–2` queries • Production Edge-Case & Pitfall Guard (Identifying and Fixing the N+1 Query Problem) • Hands-On Python Implementation & Verification (Identifying and Fixing the N+1 Query Problem) • Identifying and Fixing the N+1 Query Problem — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Identifying • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Dropping `/api/v1/employees` response latency from `480ms` (`101` queries) to `9ms` (`1` query with `joinedload`)!

What Topic Is Next → Next Topic 11.21: Alembic Database Migrations Fundamentals

11.21 Alembic Database Migrations Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `alembic init` • Configuring `env.py` (`target_metadata = Base.metadata` + `DATABASE_URL` from env) • `alembic.ini` • `alembic_version` tracking table • Production Edge-Case & Pitfall Guard (Alembic Database Migrations Fundamentals) • Hands-On Python Implementation & Verification (Alembic Database Migrations Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Alembic Database Migrations Fundamentals

Purpose: Replacing destructive `Base.metadata.create_all()` calls with repeatable, version-controlled production migrations.

What Topic Is Next → Next Topic 11.22: Evolving Models and Autogenerating Schema Migrations

11.22 Evolving Models and Autogenerating Schema Migrations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `alembic revision --autogenerate -m "add_employee_phone"` • ALWAYS reviewing autogenerated `upgrade()`/`downgrade()` scripts • Safe nullable -> backfill -> `NOT NULL` rollouts • Production Edge-Case & Pitfall Guard (Evolving Models and Autogenerating Schema Migrations) • Hands-On Python Implementation & Verification (Evolving Models and Autogenerating Schema Migrations) • Evolving Models and Autogenerating Schema Migrations — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Evolving Models

Purpose: Adding a new column to a non-empty production table safely without locking or failing on existing rows.

What Topic Is Next → Next Topic 11.23: Testing Database Repositories in Python

11.23 Testing Database Repositories in Python [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Nested SAVEPOINT transaction rollback fixtures per test (`session.begin_nested()`) • Factory helpers • Testing real constraint/integrity errors • Production Edge-Case & Pitfall Guard (Testing Database Repositories in Python) • Hands-On Python Implementation & Verification (Testing Database Repositories in Python) • Testing Database Repositories in Python — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Testing Database Repositories in Python

Purpose: Running 100 fast, isolated repository integration tests where every test starts with a clean database state.

What Topic Is Next → Module 11 Capstone (Week 11 Lab Build: Pooled psycopg 3 & SQLAlchemy 2.0 + Alembic Repository) → Next Topic 12.1: NoSQL Introduction (Relational vs Document Databases)

Module 11 Hands-On Capstone Project (100 Marks): Week 11 Lab Build: Pooled psycopg 3 & SQLAlchemy 2.0 + Alembic Repository — Build parameterized `psycopg` 3 DAOs and modern SQLAlchemy 2.0 `Mapped[...]` repositories eliminating N+1 queries with `joinedload`/`selectinload` and managing schema versions cleanly via `A...

MODULE 12 OF 24 • STAGE 3 • 25 LIVE + LAB HOURS • 18 TOPICS (144 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 12: MongoDB and NoSQL Fundamentals

Understand document databases cleanly alongside PostgreSQL and deliver Major Project 3: NoSQL vs RDBMS trade-offs, MongoDB architecture, BSON format, database/collection ops, CRUD (`insertOne`/`find`/`updateOne`/`deleteOne`), query operators, field projection, sort/pagination, embedded documents vs references, indexes, Aggregation Pipeline, `$lookup`, transactions, `PyMongo`, async `Motor` fundamentals, schema modeling, SQL vs NoSQL selection, and Major Project 3 (`Database-Driven Employee Management Backend`).

Objectives: ✓ Explain MongoDB document/collection architecture, BSON types (`ObjectId`, `ISODate`, `Decimal128`), and NoSQL vs PostgreSQL trade-offs. • ✓ Perform MongoDB CRUD, comparison/logical filtering, field projection, sorting, pagination, and compound indexing. • ✓ Build multi-stage Aggregation Pipelines (`$match`, `$group`, `$project`, `$sort`, `$lookup`) for analytical summaries. • ✓ Connect synchronous Python apps via `PyMongo` and understand asynchronous `Motor` integration for FastAPI. • ✓ Deliver Major Project 3: a complete Database-Driven Employee Management Backend in Python + PostgreSQL + SQLAlchemy 2.0 + Alembic.

12.1 NoSQL Introduction (Relational vs Document Databases) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Document • Key-Value (Redis) • Wide-Column • Graph NoSQL families vs relational PostgreSQL tables & PostgreSQL `JSONB` • Production Edge-Case & Pitfall Guard (NoSQL Introduction (Relational vs Document Databases)) • Hands-On Python Implementation & Verification (NoSQL Introduction (Relational vs Document Databases)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for NoSQL Introduction (Relational vs Document Databases...

Purpose: Storing variable-attribute e-commerce specs (laptop RAM/CPU vs shirt size/fabric) or audit event logs.

What Topic Is Next → Next Topic 12.2: MongoDB Architecture (Databases, Collections, Documents)

12.2 MongoDB Architecture (Databases, Collections, Documents) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Databases -> Collections -> JSON-like Documents • Replica Sets (high availability) • Sharding (horizontal partition keys) overview • Production Edge-Case & Pitfall Guard (MongoDB Architecture (Databases, Collections, Documents)) • Hands-On Python Implementation & Verification (MongoDB Architecture (Databases, Collections, Documents)) • MongoDB Architecture (Databases, Collections, Documents) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Architecture (Databases, Collections, Docume...

Purpose: Understanding primary/secondary replica failover and collection-level JSON Schema validation.

What Topic Is Next → Next Topic 12.3: BSON Document Storage Format

12.3 BSON Document Storage Format [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Binary JSON (`BSON`) • 12-byte `ObjectId` (timestamp + random + counter) • `Date` • `Decimal128` for money • Production Edge-Case & Pitfall Guard (BSON Document Storage Format) • Hands-On Python Implementation & Verification (BSON Document Storage Format) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for BSON Document Storage Format

Purpose: Using `Decimal128` for prices in MongoDB and understanding why unbounded array growth hits the `16 MB` BSON limit!

What Topic Is Next → Next Topic 12.4: Database and Collection Operations

12.4 Database and Collection Operations [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lazy database/collection creation on first write • Explicit `create_collection` with `$jsonSchema` validator rules • `drop()` • Production Edge-Case & Pitfall Guard (Database and Collection Operations) • Hands-On Python Implementation & Verification (Database and Collection Operations) • Database and Collection Operations — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database

Purpose: Attaching `$jsonSchema` required-field rules to `audit_events` collections so malformed documents are rejected.

What Topic Is Next → Next Topic 12.5: MongoDB CRUD (insertOne, find, updateOne, deleteOne)

12.5 MongoDB CRUD (insertOne, find, updateOne, deleteOne) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `insert_one` / `insert_many` • `find_one` / `find` • Atomic `$set` / `$inc` / `$push` in `update_one` • `delete_one` / `delete_many` • Production Edge-Case & Pitfall Guard (MongoDB CRUD (insertOne, find, updateOne, deleteOne)) • Hands-On Python Implementation & Verification (MongoDB CRUD (insertOne, find, updateOne, deleteOne)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB CRUD (insertOne, find, updateOne, deleteOne)

Purpose: Incrementing job view counters atomically in-place via `update_one({"_id": jid}, {"$inc": {"views": 1}})`.

What Topic Is Next → Next Topic 12.6: Comparison and Logical Query Operators

12.6 Comparison and Logical Query Operators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `$eq` • `$ne` • `$gt` • `$gte` • Production Edge-Case & Pitfall Guard (Comparison and Logical Query Operators) • Hands-On Python Implementation & Verification (Comparison and Logical Query Operators) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Comparison

Purpose: Querying candidate profiles possessing ALL required skills via `{"skills": {"$all": ["python", "fastapi"]}}`.

What Topic Is Next → Next Topic 12.7: Field Projection

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

Subtopics Covered: Inclusion projection `{"title": 1 • "company": 1 • "_id": 0}` vs exclusion `{"raw_resume_text": 0}` to cut network payload size • Production Edge-Case & Pitfall Guard (Field Projection) • Hands-On Python Implementation & Verification (Field Projection) • Field Projection — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Field Projection

Purpose: Omitting 100KB parsed resume blobs when listing 20 compact job application cards.

What Topic Is Next → Next Topic 12.8: Sorting, Skip and Limit Pagination

12.8 Sorting, Skip and Limit Pagination [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.sort([("created_at" • -1) • ("_id" • -1)]).skip(offset).limit(page_size)` vs range-based `_id < last_id` cursor pagination • Production Edge-Case & Pitfall Guard (Sorting, Skip and Limit Pagination) • Hands-On Python Implementation & Verification (Sorting, Skip and Limit Pagination) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Sorting

Purpose: Paginating audit event feeds efficiently using indexed `(created_at, _id)` cursor bounds.

What Topic Is Next → Next Topic 12.9: Embedded Nested Documents

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

Subtopics Covered: Co-locating `1:1` • Bounded `1:few` data (`company.headquarters_address` • `order.shipping_snapshot`) for single-read atomicity • Production Edge-Case & Pitfall Guard (Embedded Nested Documents) • Hands-On Python Implementation & Verification (Embedded Nested Documents) • Embedded Nested Documents — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Embedded Nested Documents

Purpose: Embedding immutable `shipping_address` snapshot directly inside an `Order` document so later profile edits never alter past orders!

What Topic Is Next → Next Topic 12.10: Document References

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

Subtopics Covered: Storing `employer_id` / `user_id` references across separate collections for `1:many` (unbounded) • `many:many` relationships • Production Edge-Case & Pitfall Guard (Document References) • Hands-On Python Implementation & Verification (Document References) • Document References — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Document References • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Keeping `job_applications` as its own collection referencing `job_id` and `candidate_id` because a job can receive 50,000 applications!

What Topic Is Next → Next Topic 12.11: MongoDB Indexing

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

Subtopics Covered: Single-field • Compound `(tenant_id: 1 • Created_at: -1)` (ESR rule: Equality • Sort • Production Edge-Case & Pitfall Guard (MongoDB Indexing) • Hands-On Python Implementation & Verification (MongoDB Indexing) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for MongoDB Indexing

Purpose: Auto-expiring OTP and webhook deduplication documents after 24 hours with zero cron code using a MongoDB TTL index!

What Topic Is Next → Next Topic 12.12: Aggregation Pipeline ($match, $group, $project, $sort)

12.12 Aggregation Pipeline ($match, $group, $project, $sort) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Multi-stage pipeline `[ {"$match": ...} • {"$group": ...} • {"$sort": ...} • {"$limit": ...} ]` placing indexed `$match` FIRST • Production Edge-Case & Pitfall Guard (Aggregation Pipeline ($match, $group, $project, $sort)) • Hands-On Python Implementation & Verification (Aggregation Pipeline ($match, $group, $project, $sort)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Aggregation Pipeline ($match, $group, $project, $sor...

Purpose: Aggregating event counts by event type and status directly inside MongoDB instead of pulling 100k docs into Python.

What Topic Is Next → Next Topic 12.13: $lookup Cross-Collection Joins

12.13 $lookup Cross-Collection Joins [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Left-outer collection join via `{"$lookup": {"from": ... • "localField": ... • "foreignField": ... • "as": ...}}` • Production Edge-Case & Pitfall Guard ($lookup Cross-Collection Joins) • Hands-On Python Implementation & Verification ($lookup Cross-Collection Joins) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for $lookup Cross-Collection Joins

Purpose: Enriching a paginated page of 20 applications with employer summary metadata.

What Topic Is Next → Next Topic 12.14: Multi-Document Transactions Fundamentals

12.14 Multi-Document Transactions Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Replica-set `with client.start_session() as s: with s.start_transaction():` vs single-document atomicity design • Production Edge-Case & Pitfall Guard (Multi-Document Transactions Fundamentals) • Hands-On Python Implementation & Verification (Multi-Document Transactions Fundamentals) • Multi-Document Transactions Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multi-Document Transactions Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Knowing MongoDB supports ACID multi-document transactions on replica sets while single-document atomic design remains fastest.

What Topic Is Next → Next Topic 12.15: Python Integration with PyMongo

12.15 Python Integration with PyMongo [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from pymongo import MongoClient • ASCENDING • DESCENDING` • Singleton `MongoClient` thread-safe pooling • Production Edge-Case & Pitfall Guard (Python Integration with PyMongo) • Hands-On Python Implementation & Verification (Python Integration with PyMongo) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Python Integration with PyMongo

Purpose: Writing audit events and product catalog documents from synchronous Python/Django services via a shared `MongoClient`.

What Topic Is Next → Next Topic 12.16: Asynchronous MongoDB Driver Fundamentals (Motor)

12.16 Asynchronous MongoDB Driver Fundamentals (Motor) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `motor.motor_asyncio.AsyncIOMotorClient` (• PyMongo async API) for non-blocking `await collection.find_one(...)` inside FastAPI `async def` routes • Production Edge-Case & Pitfall Guard (Asynchronous MongoDB Driver Fundamentals (Motor)) • Hands-On Python Implementation & Verification (Asynchronous MongoDB Driver Fundamentals (Motor)) • Asynchronous MongoDB Driver Fundamentals (Motor) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Asynchronous MongoDB Driver Fundamentals (Motor) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Serving non-blocking async catalog reads inside FastAPI without stalling the Uvicorn event loop.

What Topic Is Next → Next Topic 12.17: Data Modelling (Embedding vs Referencing)

12.17 Data Modelling (Embedding vs Referencing) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Practical decision checklist: cardinality (`1:few` bounded -> embed • `1:many` unbounded -> reference) • Read/write ratio • Independent access patterns • Production Edge-Case & Pitfall Guard (Data Modelling (Embedding vs Referencing)) • Hands-On Python Implementation & Verification (Data Modelling (Embedding vs Referencing)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Data Modelling (Embedding vs Referencing)

Purpose: Embedding `skills: ["Python", "SQL"]` (`<= 30` items) inside `Candidate` while referencing `Application` documents separately.

What Topic Is Next → Next Topic 12.18: Selecting PostgreSQL SQL vs MongoDB NoSQL

12.18 Selecting PostgreSQL SQL vs MongoDB NoSQL [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Polyglot persistence architecture: PostgreSQL (users • Auth • Orders • Wallets • Production Edge-Case & Pitfall Guard (Selecting PostgreSQL SQL vs MongoDB NoSQL) • Hands-On Python Implementation & Verification (Selecting PostgreSQL SQL vs MongoDB NoSQL) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Selecting PostgreSQL SQL vs MongoDB NoSQL

Purpose: Defending clear database choices in backend system-design interviews and delivering Major Project 3 (`Database-Driven Employee Management Backend`) on PostgreSQL + SQLAlc...

What Topic Is Next → Module 12 Capstone (Project 3: Database-Driven Employee Management Backend) → Next Topic 13.1: Django Framework Architecture (MTV)

Module 12 Hands-On Capstone Project (100 Marks): Project 3: Database-Driven Employee Management Backend — Engineer a production-structured relational Employee, Department & Salary Management Backend in Python 3.12 backed by PostgreSQL 16, SQLAlchemy 2.0 ORM, and Alembic schema migrations — featu...

MODULE 13 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 26 TOPICS (208 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 13: Django Fundamentals and Architecture

Master Django 5 architecture for structured, database-driven business backends: MTV architecture, Django vs Flask vs FastAPI framework strategy, `venv` setup, `django-admin` & `startapp`, `settings`/`urls`/`wsgi`/`asgi`, environment settings, URL routing, FBVs & CBVs, `HttpRequest`/`HttpResponse`, templates & inheritance, static/media files, Forms & validation lifecycle, Models, fields, `makemigrations`/`migrate`, Django Admin, ORM QuerySets, CRUD, `Q`/`F` filtering, `ForeignKey`/`OneToOneField`/`ManyToManyField`, and `select_related`/`prefetch_related`.

Objectives: ✓ Explain Django 5 Model-Template-View (MTV) request flow and compare Django ("batteries-included" RDBMS apps) vs FastAPI (async APIs/microservices) vs Flask (minimalist microframework). • ✓ Bootstrap clean multi-app Django projects (`config/`, `apps/`) with environment-driven `settings.py`, URL namespaces, FBVs, and CBVs. • ✓ Design PostgreSQL-backed Django `Model` classes, generate & inspect `makemigrations`/`migrate` revisions, and customize `ModelAdmin`. • ✓ Write expressive, lazy Django ORM `QuerySet` chains using `filter`, `exclude`, `get`, `Q()`, `F()`, annotations, and relational fields. • ✓ Eliminate Django ORM `N+1` queries proactively using `select_related()` (SQL JOIN) and `prefetch_related()` (batched `IN` query).

13.1 Django Framework Architecture (MTV) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Model-Template-View (MTV) mapping to MVC • URL dispatcher -> Middleware -> View -> ORM Model -> Response lifecycle • Production Edge-Case & Pitfall Guard (Django Framework Architecture (MTV)) • Hands-On Python Implementation & Verification (Django Framework Architecture (MTV)) • Django Framework Architecture (MTV) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Framework Architecture (MTV) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Tracing an incoming HTTP request through Django middleware, URL resolver, view logic, PostgreSQL ORM, and JSON/HTML response.

What Topic Is Next → Next Topic 13.2: Django vs Flask vs FastAPI Comparison

13.2 Django vs Flask vs FastAPI Comparison [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Batteries-included Django + DRF (ORM • Admin • Auth • Migrations) vs ASGI type-driven FastAPI (Pydantic • Production Edge-Case & Pitfall Guard (Django vs Flask vs FastAPI Comparison) • Hands-On Python Implementation & Verification (Django vs Flask vs FastAPI Comparison) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django vs Flask vs FastAPI Comparison

Purpose: Choosing Django + DRF for multi-module business portals with rich Admin/RBAC and FastAPI for high-throughput async APIs/microservices (while understanding Flask micro-app...

What Topic Is Next → Next Topic 13.3: Python Virtual Environment Setup for Django

13.3 Python Virtual Environment Setup for Django [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Isolated `.venv` • Installing `Django>=5.1 • <5.2` + `psycopg[binary]` + `python-dotenv` • Freezing deterministic dependencies • Production Edge-Case & Pitfall Guard (Python Virtual Environment Setup for Django) • Hands-On Python Implementation & Verification (Python Virtual Environment Setup for Django) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Python Virtual Environment Setup for Django

Purpose: Ensuring clean, reproducible Django 5 + PostgreSQL environments across developer laptops and CI.

What Topic Is Next → Next Topic 13.4: Django Installation and Version Management

13.4 Django Installation and Version Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Django LTS vs feature release cadence • `python -m django --version` • Security patch upgrades • Compatibility with Python 3.12 • Production Edge-Case & Pitfall Guard (Django Installation and Version Management) • Hands-On Python Implementation & Verification (Django Installation and Version Management) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Installation

Purpose: Tracking Django LTS / 5.x security advisories so production dependencies stay patched.

What Topic Is Next → Next Topic 13.5: Project Creation with django-admin

13.5 Project Creation with django-admin [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `django-admin startproject config .` clean root-level `manage.py` layout (avoiding double-nested `project/project/` folders!) • Production Edge-Case & Pitfall Guard (Project Creation with django-admin) • Hands-On Python Implementation & Verification (Project Creation with django-admin) • Project Creation with django-admin — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Project Creation with django-admin • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Starting every professional Django repo with `django-admin startproject config .` so `manage.py` sits cleanly at repo root.

What Topic Is Next → Next Topic 13.6: Application Creation with startapp

13.6 Application Creation with startapp [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `python manage.py startapp jobs` • Cohesive bounded Django apps (`accounts` • `companies` • `jobs` • Production Edge-Case & Pitfall Guard (Application Creation with startapp) • Hands-On Python Implementation & Verification (Application Creation with startapp) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Application Creation with startapp

Purpose: Decomposing a Job Portal into focused, reusable feature apps rather than one giant `core` app.

What Topic Is Next → Next Topic 13.7: Django Project Structure (settings, urls, wsgi, asgi)

13.7 Django Project Structure (settings, urls, wsgi, asgi) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Responsibilities of `manage.py` • `config/settings.py` • Root `config/urls.py` • Synchronous `wsgi.py` (Gunicorn) • Production Edge-Case & Pitfall Guard (Django Project Structure (settings, urls, wsgi, asgi)) • Hands-On Python Implementation & Verification (Django Project Structure (settings, urls, wsgi, asgi)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Project Structure (settings, urls, wsgi, asgi...

Purpose: Knowing how Gunicorn loads `config.wsgi:application` vs Uvicorn loading `config.asgi:application`.

What Topic Is Next → Next Topic 13.8: Environment-Specific Settings Configuration

13.8 Environment-Specific Settings Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Loading `SECRET_KEY` • `DEBUG` • `ALLOWED_HOSTS` • PostgreSQL `DATABASES` strictly from environment variables (`os.environ`) • Production Edge-Case & Pitfall Guard (Environment-Specific Settings Configuration) • Hands-On Python Implementation & Verification (Environment-Specific Settings Configuration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment-Specific Settings Configuration

Purpose: Guaranteeing `DEBUG = False` and strict `ALLOWED_HOSTS` in production without changing source files.

What Topic Is Next → Next Topic 13.9: URL Routing and Path Converters

13.9 URL Routing and Path Converters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `path()` • `include()` app URL delegation • `app_name` namespaces • Typed converters (`<int:pk>` • Production Edge-Case & Pitfall Guard (URL Routing and Path Converters) • Hands-On Python Implementation & Verification (URL Routing and Path Converters) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for URL Routing

Purpose: Delegating `/api/v1/jobs/` cleanly via `include(("jobs.urls", "jobs"), namespace="jobs")` and generating URLs with `reverse("jobs:detail", args=[pk])`.

What Topic Is Next → Next Topic 13.10: Function-Based Views (FBVs)

13.10 Function-Based Views (FBVs) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit `def view(request • ...):` handlers • `@require_http_methods(["GET" • "POST"])` • Production Edge-Case & Pitfall Guard (Function-Based Views (FBVs)) • Hands-On Python Implementation & Verification (Function-Based Views (FBVs)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Function-Based Views (FBVs)

Purpose: Writing crisp, readable `/healthz` readiness endpoints and custom webhook receivers.

What Topic Is Next → Next Topic 13.11: Class-Based Views (CBVs) Fundamentals

13.11 Class-Based Views (CBVs) Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `django.views.View` • HTTP method dispatch (`.get()` • `.post()`) • `View.as_view()` • Production Edge-Case & Pitfall Guard (Class-Based Views (CBVs) Fundamentals) • Hands-On Python Implementation & Verification (Class-Based Views (CBVs) Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Class-Based Views (CBVs) Fundamentals

Purpose: Understanding `as_view()` and `dispatch()` mechanics before building DRF `APIView` and `ModelViewSet` classes in Week 14!

What Topic Is Next → Next Topic 13.12: HttpRequest and HttpResponse Lifecycle

13.12 HttpRequest and HttpResponse Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `request.method` • `request.GET` (`QueryDict`) • `request.POST` • `request.body` • Production Edge-Case & Pitfall Guard (HttpRequest and HttpResponse Lifecycle) • Hands-On Python Implementation & Verification (HttpRequest and HttpResponse Lifecycle) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HttpRequest

Purpose: Reading `request.headers.get("X-Request-Id")` and query lists `request.GET.getlist("skill")` accurately.

What Topic Is Next → Next Topic 13.13: Django Template Engine Fundamentals

13.13 Django Template Engine Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: DTL context variables `{{ var }}` • Auto-escaping XSS protection • Filters (`|date` • `|default`) • Production Edge-Case & Pitfall Guard (Django Template Engine Fundamentals) • Hands-On Python Implementation & Verification (Django Template Engine Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Template Engine Fundamentals

Purpose: Rendering transactional HTML emails (welcome, password reset, job alert) and internal admin portals safely.

What Topic Is Next → Next Topic 13.14: Template Inheritance and Base Layouts

13.14 Template Inheritance and Base Layouts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `base.html` master skeleton • `{% block content %}` • `{% extends "base.html" %}` • Reusable `{% include %}` partials • Production Edge-Case & Pitfall Guard (Template Inheritance and Base Layouts) • Hands-On Python Implementation & Verification (Template Inheritance and Base Layouts) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Template Inheritance

Purpose: Sharing consistent header/footer email branding across Registration, Password Reset, and Interview Invite emails.

What Topic Is Next → Next Topic 13.15: Static and Media Files Configuration

13.15 Static and Media Files Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `STATIC_URL` • `STATIC_ROOT` • `collectstatic` (WhiteNoise/Nginx/CDN) vs user-uploaded `MEDIA_URL` / `MEDIA_ROOT` / S3 `STORAGES` • Production Edge-Case & Pitfall Guard (Static and Media Files Configuration) • Hands-On Python Implementation & Verification (Static and Media Files Configuration) • Static and Media Files Configuration — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Static

Purpose: Serving Django Admin/Swagger CSS via `collectstatic` while isolating user-uploaded PDF resumes in `MEDIA_ROOT` / S3.

What Topic Is Next → Next Topic 13.16: Django Forms

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

Subtopics Covered: `forms.Form` vs `forms.ModelForm` • Explicit `fields = [...]` allowlists • `{% csrf_token %}` • `form.is_valid()` + `cleaned_data` • Production Edge-Case & Pitfall Guard (Django Forms) • Hands-On Python Implementation & Verification (Django Forms) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Forms

Purpose: Understanding Django's `cleaned_data` validation pipeline which directly inspires DRF `Serializer` validation!

What Topic Is Next → Next Topic 13.17: Form Validation Lifecycle (clean and clean_<field>)

13.17 Form Validation Lifecycle (clean and clean_<field>) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Field validators -> `clean_<fieldname>()` single-field normalization -> `clean()` cross-field validation (`max_salary >= min_salary`) -> `ValidationError` • Production Edge-Case & Pitfall Guard (Form Validation Lifecycle (clean and clean_<field>)) • Hands-On Python Implementation & Verification (Form Validation Lifecycle (clean and clean_<field>)) • Form Validation Lifecycle (clean and clean_<field>) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Form Validation Lifecycle (clean and clean_<field>) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Enforcing cross-field business rules (`closing_date > today` and `max_salary >= min_salary`) with structured field errors.

What Topic Is Next → Next Topic 13.18: Django Models Introduction

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

Subtopics Covered: `models.Model` subclasses • Implicit `BigAutoField` PK (`DEFAULT_AUTO_FIELD`) • `class Meta` (`db_table` • `ordering` • Production Edge-Case & Pitfall Guard (Django Models Introduction) • Hands-On Python Implementation & Verification (Django Models Introduction) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Models Introduction

Purpose: Defining `Company` and `JobPosting` models with database-enforced `CheckConstraint` and `UniqueConstraint` in `Meta`!

What Topic Is Next → Next Topic 13.19: Model Field Types and Options

13.19 Model Field Types and Options [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CharField` • `SlugField` • `TextField` • `DecimalField(max_digits=12 • Production Edge-Case & Pitfall Guard (Model Field Types and Options) • Hands-On Python Implementation & Verification (Model Field Types and Options) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Model Field Types

Purpose: Using `models.TextChoices` for job statuses and `DecimalField` for CTC figures.

What Topic Is Next → Next Topic 13.20: Schema Migrations (makemigrations and migrate)

13.20 Schema Migrations (makemigrations and migrate) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `makemigrations` • `sqlmigrate <app> <num>` (inspecting exact SQL!) • `migrate` • `showmigrations` • Production Edge-Case & Pitfall Guard (Schema Migrations (makemigrations and migrate)) • Hands-On Python Implementation & Verification (Schema Migrations (makemigrations and migrate)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Schema Migrations (makemigrations and migrate)

Purpose: Running `python manage.py sqlmigrate jobs 0001` before every PR to verify exact PostgreSQL DDL and index locks!

What Topic Is Next → Next Topic 13.21: Django Admin Panel Customization

13.21 Django Admin Panel Customization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@admin.register(JobPosting)` • `list_display` • `list_filter` • `search_fields` • Production Edge-Case & Pitfall Guard (Django Admin Panel Customization) • Hands-On Python Implementation & Verification (Django Admin Panel Customization) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Admin Panel Customization

Purpose: Giving support/operations teams an instant, N+1-free internal moderation dashboard in 25 lines of Python.

What Topic Is Next → Next Topic 13.22: Django ORM and QuerySet Evaluation

13.22 Django ORM and QuerySet Evaluation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lazy `QuerySet` chaining • Exact evaluation triggers (iteration • Slicing with step • `len()` • Production Edge-Case & Pitfall Guard (Django ORM and QuerySet Evaluation) • Hands-On Python Implementation & Verification (Django ORM and QuerySet Evaluation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django ORM

Purpose: Using `qs.exists()` (`SELECT 1 ... LIMIT 1`) instead of `if qs:` (which loads ALL matching model instances into Python memory!).

What Topic Is Next → Next Topic 13.23: ORM CRUD Operations

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

Subtopics Covered: `objects.create()` • `bulk_create()` • `get_or_create()` • `update_or_create()` • Production Edge-Case & Pitfall Guard (ORM CRUD Operations) • Hands-On Python Implementation & Verification (ORM CRUD Operations) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ORM CRUD Operations

Purpose: Passing `save(update_fields=["status", "updated_at"])` so concurrent updates to other columns on the same row are never overwritten!

What Topic Is Next → Next Topic 13.24: Filtering QuerySets (filter, exclude, get, Q, F)

13.24 Filtering QuerySets (filter, exclude, get, Q, F) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Field lookups (`__iexact` • `__icontains` • `__in` • `__gte` • Production Edge-Case & Pitfall Guard (Filtering QuerySets (filter, exclude, get, Q, F)) • Hands-On Python Implementation & Verification (Filtering QuerySets (filter, exclude, get, Q, F)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Filtering QuerySets (filter, exclude, get, Q, F)

Purpose: Incrementing view counts atomically in SQL via `Job.objects.filter(pk=pk).update(views=F("views") + 1)` with zero race conditions!

What Topic Is Next → Next Topic 13.25: Model Relationships (ForeignKey, OneToOneField, ManyToManyField)

13.25 Model Relationships (ForeignKey, OneToOneField, ManyToManyField) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit `on_delete=models.CASCADE / PROTECT / SET_NULL` • Explicit `related_name="jobs"` • `OneToOneField` (`CandidateProfile`) • `ManyToManyField(... • Production Edge-Case & Pitfall Guard (Model Relationships (ForeignKey, OneToOneField, ManyToManyField)) • Hands-On Python Implementation & Verification (Model Relationships (ForeignKey, OneToOneField, ManyToManyField)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Model Relationships (ForeignKey, OneToOneField, Many...

Purpose: Using `on_delete=models.PROTECT` on financial/employer references and explicit `through="JobApplication"` models storing `status` + `applied_at`.

What Topic Is Next → Next Topic 13.26: Django Query Optimization (select_related and prefetch_related)

13.26 Django Query Optimization (select_related and prefetch_related) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `select_related("company" • "company__owner")` (`SQL INNER/LEFT JOIN` for FK/OneToOne) vs `prefetch_related("skills" • Prefetch(...))` (separate `IN` query for M2M/reverse FK) • `.only()`/`.values()` • Production Edge-Case & Pitfall Guard (Django Query Optimization (select_related and prefetch_related)) • Hands-On Python Implementation & Verification (Django Query Optimization (select_related and prefetch_related)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Query Optimization (select_related and prefet...

Purpose: Fetching 25 jobs with company details AND all required skill tags in exactly 2 SQL queries instead of `1 + 25 + 25 = 51` queries!

What Topic Is Next → Module 13 Capstone (Week 13 Lab Build: Modular Django 5 Recruitment Core & Optimized Admin) → Next Topic 14.1: REST Architecture Principles

Module 13 Hands-On Capstone Project (100 Marks): Week 13 Lab Build: Modular Django 5 Recruitment Core & Optimized Admin — Bootstrap a multi-app Django 5 project (`accounts`, `companies`, `jobs`) backed by PostgreSQL with environment-driven settings, `CheckConstraint` models, customized N+1-free `ModelAdmin`, an...

MODULE 14 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 14: Django Advanced Development and REST API Fundamentals

Design clean, standards-compliant RESTful JSON APIs with Django REST Framework (DRF): REST principles, HTTP methods (`GET`/`POST`/`PUT`/`PATCH`/`DELETE`), status codes (`2xx`–`5xx`), headers, JSON contracts, DRF setup, `Serializer` & `ModelSerializer`, field & object validation, `APIView`, `GenericAPIView`, Mixins, `ModelViewSet`, Routers, CRUD APIs, `django-filter`, `SearchFilter`, `OrderingFilter`, Pagination, Permissions,

Throttling, Versioning, standardized responses, and custom exception handlers.

Objectives: ✓ Design resource-oriented REST URLs (`/api/v1/jobs`, `/api/v1/jobs/{id}/applications`) using accurate HTTP methods and status codes. • ✓ Validate incoming payloads and serialize ORM models cleanly with DRF `Serializer`, `ModelSerializer`, `validate_<field>`, and `validate()`. • ✓ Progress systematically from `APIView` -> `GenericAPIView` + Mixins -> `ModelViewSet` + `DefaultRouter` knowing when each fits best. • ✓ Equip list endpoints with `django-filter`, `SearchFilter`, `OrderingFilter`, bounded pagination, permissions, rate throttling, and URL versioning. • ✓ Standardize JSON error payloads globally via custom `EXCEPTION_HANDLER` configuration.

14.1 REST Architecture Principles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Statelessness • Resource-noun URLs (`/jobs` • NOT `/getAllJobs`) • Uniform interface • Production Edge-Case & Pitfall Guard (REST Architecture Principles) • Hands-On Python Implementation & Verification (REST Architecture Principles) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST Architecture Principles

Purpose: Designing clean `/api/v1/companies/{id}/jobs` REST contracts frontend React/Angular/mobile teams love consuming.

What Topic Is Next → Next Topic 14.2: HTTP Methods (GET, POST, PUT, PATCH, DELETE)

14.2 HTTP Methods (GET, POST, PUT, PATCH, DELETE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Safe (`GET` • `HEAD` • `OPTIONS`) vs Idempotent (`GET` • `PUT` • Production Edge-Case & Pitfall Guard (HTTP Methods (GET, POST, PUT, PATCH, DELETE)) • Hands-On Python Implementation & Verification (HTTP Methods (GET, POST, PUT, PATCH, DELETE)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Methods (GET, POST, PUT, PATCH, DELETE)

Purpose: Using `PATCH /api/v1/applications/{id}` to update `{"status": "shortlisted"}` without requiring the client to resend all 15 fields!

What Topic Is Next → Next Topic 14.3: HTTP Status Codes (2xx, 3xx, 4xx, 5xx)

14.3 HTTP Status Codes (2xx, 3xx, 4xx, 5xx) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `200 OK` • `201 Created` (+ `Location`) • `204 No Content` • `400 Bad Request` • Production Edge-Case & Pitfall Guard (HTTP Status Codes (2xx, 3xx, 4xx, 5xx)) • Hands-On Python Implementation & Verification (HTTP Status Codes (2xx, 3xx, 4xx, 5xx)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Status Codes (2xx, 3xx, 4xx, 5xx)

Purpose: Returning `201 Created` on `POST`, `204 No Content` on `DELETE`, `401` for missing/expired JWT, `403` for wrong role, and `409` for duplicate job application.

What Topic Is Next → Next Topic 14.4: HTTP Request Headers (Content-Type, Accept, Authorization)

14.4 HTTP Request Headers (Content-Type, Accept, Authorization) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Content-Type: application/json` vs `multipart/form-data` • `Accept` • `Authorization: Bearer <jwt>` • `X-Request-ID` • Production Edge-Case & Pitfall Guard (HTTP Request Headers (Content-Type, Accept, Authorization)) • Hands-On Python Implementation & Verification (HTTP Request Headers (Content-Type, Accept, Authorization)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Request Headers (Content-Type, Accept, Authoriz...

Purpose: Parsing JSON bodies, negotiating API media types, and propagating correlation IDs across logs.

What Topic Is Next → Next Topic 14.5: JSON Request and Response Payloads

14.5 JSON Request and Response Payloads [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Consistent `snake_case` (or camelCase adapter) field naming • ISO-8601 UTC timestamps (`2026-11-18T06:30:00Z`) • Stringified `Decimal` money (`"85000.00"`) • Nullability • Production Edge-Case & Pitfall Guard (JSON Request and Response Payloads) • Hands-On Python Implementation & Verification (JSON Request and Response Payloads) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JSON Request

Purpose: Serializing `DecimalField` as exact strings or integer minor units so JavaScript `Number` never loses precision.

What Topic Is Next → Next Topic 14.6: Django REST Framework (DRF) Installation and Architecture

14.6 Django REST Framework (DRF) Installation and Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `djangorestframework` • `REST_FRAMEWORK` central settings dict • DRF `Request` (`request.data` • `request.query_params`) & `Response` content negotiation • Production Edge-Case & Pitfall Guard (Django REST Framework (DRF) Installation and Architecture) • Hands-On Python Implementation & Verification (Django REST Framework (DRF) Installation and Architecture) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django REST Framework (DRF) Installation

Purpose: Configuring secure project-wide defaults (`DEFAULT_PERMISSION_CLASSES = ["IsAuthenticated"]`, `DEFAULT_PAGINATION_CLASS`) in `settings.py`!

What Topic Is Next → Next Topic 14.7: DRF Serializers Fundamentals

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

Subtopics Covered: `serializers.Serializer` for non-model inputs (login • Password reset • Search filters) • `is_valid(raise_exception=True)` • Production Edge-Case & Pitfall Guard (DRF Serializers Fundamentals) • Hands-On Python Implementation & Verification (DRF Serializers Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DRF Serializers Fundamentals

Purpose: Validating complex non-ORM command payloads like `LoginSerializer` or `BulkInviteSerializer`.

What Topic Is Next → Next Topic 14.8: ModelSerializer

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

Subtopics Covered: `serializers.ModelSerializer` • Explicit `fields` + `read_only_fields` (`id` • `employer` • `created_at`) • Production Edge-Case & Pitfall Guard (ModelSerializer) • Hands-On Python Implementation & Verification (ModelSerializer) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ModelSerializer

Purpose: Preventing candidates/employers from spoofing `employer_id` or `is_verified` via `read_only_fields` + `serializer.save(employer=request.user.company)`.

What Topic Is Next → Next Topic 14.9: Field-Level and Object-Level Serializer Validation

14.9 Field-Level and Object-Level Serializer Validation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `validate_<field>(self • Value)` single-field rules • `validate(self • Attrs)` cross-field checks (handling partial `PATCH` with `self.instance`!) • Production Edge-Case & Pitfall Guard (Field-Level and Object-Level Serializer Validation) • Hands-On Python Implementation & Verification (Field-Level and Object-Level Serializer Validation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Field-Level

Purpose: Validating `max_salary >= min_salary` across both `POST` (full attrs) and `PATCH` (merging `attrs` with `self.instance`!).

What Topic Is Next → Next Topic 14.10: DRF APIView

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

Subtopics Covered: `rest_framework.views.APIView` (auth -> permission -> throttle pipeline BEFORE `get`/`post`/`patch`/`delete` runs) • `@api_view` decorator • Production Edge-Case & Pitfall Guard (DRF APIView) • Hands-On Python Implementation & Verification (DRF APIView) • DRF APIView — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DRF APIView • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Building custom dashboard summary, webhook, and authentication endpoints (`RegisterAPIView`, `MeAPIView`) that don't map 1:1 to CRUD.

What Topic Is Next → Next Topic 14.11: GenericAPIView and Concrete Generic Views

14.11 GenericAPIView and Concrete Generic Views [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `GenericAPIView` (`get_queryset()` • `get_serializer_class()` • `lookup_field`) • Concrete `ListCreateAPIView` • Production Edge-Case & Pitfall Guard (GenericAPIView and Concrete Generic Views) • Hands-On Python Implementation & Verification (GenericAPIView and Concrete Generic Views) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GenericAPIView

Purpose: Scoping `def get_queryset(self):` dynamically to `JobApplication.objects.filter(candidate=self.request.user)` for tenant isolation!

What Topic Is Next → Next Topic 14.12: DRF Mixins for Reusable API Behaviour

14.12 DRF Mixins for Reusable API Behaviour [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ListModelMixin` • `CreateModelMixin` • `RetrieveModelMixin` • `UpdateModelMixin` • Production Edge-Case & Pitfall Guard (DRF Mixins for Reusable API Behaviour) • Hands-On Python Implementation & Verification (DRF Mixins for Reusable API Behaviour) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DRF Mixins for Reusable API Behaviour

Purpose: Composing a Create + List + Retrieve-only audit resource (`CreateModelMixin, ListModelMixin, RetrieveModelMixin, GenericViewSet`) that physically cannot be updated or del...

What Topic Is Next → Next Topic 14.13: ViewSets and ModelViewSet

14.13 ViewSets and ModelViewSet [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ViewSet` • `ReadOnlyModelViewSet` • `ModelViewSet` • Switching serializers per action (`if self.action == "list":`) • Production Edge-Case & Pitfall Guard (ViewSets and ModelViewSet) • Hands-On Python Implementation & Verification (ViewSets and ModelViewSet) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for ViewSets

Purpose: Adding `@action(detail=True, methods=["post"]) def publish(self, request, pk=None):` cleanly at `POST /api/v1/jobs/{id}/publish/`.

What Topic Is Next → Next Topic 14.14: DRF Routers (DefaultRouter and SimpleRouter)

14.14 DRF Routers (DefaultRouter and SimpleRouter) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SimpleRouter` vs `DefaultRouter` • `router.register("jobs" • JobViewSet • Basename="job")` • Production Edge-Case & Pitfall Guard (DRF Routers (DefaultRouter and SimpleRouter)) • Hands-On Python Implementation & Verification (DRF Routers (DefaultRouter and SimpleRouter)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DRF Routers (DefaultRouter and SimpleRouter)

Purpose: Generating consistent REST routes automatically and referencing `reverse("job-list")` / `reverse("job-detail", args=[id])` in tests.

What Topic Is Next → Next Topic 14.15: Complete CRUD REST API Endpoints

14.15 Complete CRUD REST API Endpoints [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: End-to-end `POST 201` • `GET 200` list/detail • `PATCH 200` • `PUT 200` • Production Edge-Case & Pitfall Guard (Complete CRUD REST API Endpoints) • Hands-On Python Implementation & Verification (Complete CRUD REST API Endpoints) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Complete CRUD REST API Endpoints

Purpose: Delivering complete Company & Job Posting CRUD APIs backed by PostgreSQL with `<= 2` queries per request.

What Topic Is Next → Next Topic 14.16: API Filtering with django-filter

14.16 API Filtering with django-filter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `django-filter` • `DjangoFilterBackend` • Custom `FilterSet` (`min_salary = NumberFilter(field_name="min_salary" • Lookup_expr="gte")` • Production Edge-Case & Pitfall Guard (API Filtering with django-filter) • Hands-On Python Implementation & Verification (API Filtering with django-filter) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Filtering with django-filter

Purpose: Powering `/api/v1/jobs/?is_remote=true&min_salary=800000&city=Bengaluru` multi-facet filtering cleanly.

What Topic Is Next → Next Topic 14.17: Full-Text Search with SearchFilter

14.17 Full-Text Search with SearchFilter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `filters.SearchFilter` • `search_fields = ["title" • "company__name" • "=job_code"]` (`^` prefix • Production Edge-Case & Pitfall Guard (Full-Text Search with SearchFilter) • Hands-On Python Implementation & Verification (Full-Text Search with SearchFilter) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Full-Text Search with SearchFilter

Purpose: Enabling `?search=fastapi+docker` across job titles, skill tags, and company names.

What Topic Is Next → Next Topic 14.18: Dynamic Sorting with OrderingFilter

14.18 Dynamic Sorting with OrderingFilter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `filters.OrderingFilter` • Explicit `ordering_fields = ["published_at" • "min_salary"]` allowlist • Default `ordering = ["-published_at" • Production Edge-Case & Pitfall Guard (Dynamic Sorting with OrderingFilter) • Hands-On Python Implementation & Verification (Dynamic Sorting with OrderingFilter) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Sorting with OrderingFilter

Purpose: Allowing candidates to sort `?ordering=-min_salary` or `?ordering=-published_at` safely on indexed columns.

What Topic Is Next → Next Topic 14.19: Pagination (PageNumberPagination and LimitOffsetPagination)

14.19 Pagination (PageNumberPagination and LimitOffsetPagination) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Custom `StandardResultsSetPagination(PageNumberPagination)` (`page_size = 20` • `page_size_query_param = "page_size"` • `max_page_size = 100`) vs `CursorPagination` • Production Edge-Case & Pitfall Guard (Pagination (PageNumberPagination and LimitOffsetPagination)) • Hands-On Python Implementation & Verification (Pagination (PageNumberPagination and LimitOffsetPagination)) • Pagination (PageNumberPagination and LimitOffsetPagination) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pagination (PageNumberPagination and LimitOffsetPagi...

Purpose: Capping `max_page_size = 100` so no client can request `?page_size=1000000` and crash Gunicorn workers!

What Topic Is Next → Next Topic 14.20: DRF Permission Classes

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

Subtopics Covered: `IsAuthenticated` • `IsAuthenticatedOrReadOnly` • `IsAdminUser` • Custom `BasePermission` (`has_permission` view-level vs `has_object_permission` row-level!) • Production Edge-Case & Pitfall Guard (DRF Permission Classes) • Hands-On Python Implementation & Verification (DRF Permission Classes) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DRF Permission Classes

Purpose: Enforcing `IsEmployerOrReadOnly` + `IsJobOwner` so Employer A can NEVER `PATCH` or `DELETE` Employer B's job posting (preventing IDOR / Broken Object Level Authorization)...

What Topic Is Next → Next Topic 14.21: API Rate Throttling (AnonRateThrottle, UserRateThrottle)

14.21 API Rate Throttling (AnonRateThrottle, UserRateThrottle) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AnonRateThrottle` • `UserRateThrottle` • `ScopedRateThrottle` (`login: "5/min"`) • Redis cache backing • Production Edge-Case & Pitfall Guard (API Rate Throttling (AnonRateThrottle, UserRateThrottle)) • Hands-On Python Implementation & Verification (API Rate Throttling (AnonRateThrottle, UserRateThrottle)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Rate Throttling (AnonRateThrottle, UserRateThrot...

Purpose: Throttling `/api/v1/auth/login` to `5/min` per IP to stop credential-stuffing brute-force attacks.

What Topic Is Next → Next Topic 14.22: API Versioning Strategies

14.22 API Versioning Strategies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `URLPathVersioning` (`/api/v1/jobs/`) vs `NamespaceVersioning` vs `AcceptHeaderVersioning` • `request.version` • Backward-compatible deprecation • Production Edge-Case & Pitfall Guard (API Versioning Strategies) • Hands-On Python Implementation & Verification (API Versioning Strategies) • API Versioning Strategies — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Versioning Strategies

Purpose: Evolving `/api/v2/jobs/` payload shapes cleanly without breaking live mobile/web clients still calling `/api/v1/jobs/`.

What Topic Is Next → Next Topic 14.23: Standardized JSON API Response Envelopes

14.23 Standardized JSON API Response Envelopes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Consistent pagination (`count` • `next` • `previous` • `results`) • Production Edge-Case & Pitfall Guard (Standardized JSON API Response Envelopes) • Hands-On Python Implementation & Verification (Standardized JSON API Response Envelopes) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Standardized JSON API Response Envelopes

Purpose: Ensuring frontend TypeScript Axios/Query interceptors parse every validation and domain error with one unified TypeScript interface.

What Topic Is Next → Next Topic 14.24: Custom DRF Exception Handling

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

Subtopics Covered: Writing a custom `EXCEPTION_HANDLER` wrapping DRF `exception_handler(exc • Context)` • Translating domain/Django `ValidationError` & `IntegrityError` cleanly • Never leaking stack traces • Production Edge-Case & Pitfall Guard (Custom DRF Exception Handling) • Hands-On Python Implementation & Verification (Custom DRF Exception Handling) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom DRF Exception Handling

Purpose: Catching service-layer `DuplicateApplicationError` globally and returning clean `409 Conflict` JSON with `request_id` correlation.

What Topic Is Next → Module 14 Capstone (Week 14 Lab Build: Filtered, Paginated & Versioned DRF Company/Job API) → Next Topic 15.1: Django Built-In Authentication System

Module 14 Hands-On Capstone Project (100 Marks): Week 14 Lab Build: Filtered, Paginated & Versioned DRF Company/Job API — Expose `/api/v1/companies/` and `/api/v1/jobs/` via DRF `ModelSerializer`, `ModelViewSet`, `django-filter`, `SearchFilter`, `OrderingFilter`, `max_page_size=100` pagination, object-ownership...

MODULE 15 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 28 TOPICS (224 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 15: Django Authentication, Security and Advanced APIs

Harden Django & DRF backends to industry security standards: built-in auth, custom `AbstractUser` (email login!), registration/login APIs, Argon2/PBKDF2 password hashing, password reset workflow, Session vs Token vs Stateless JWT (`djangorestframework-simplejwt`), access/refresh rotation & blacklisting, RBAC & groups, CORS, CSRF, XSS, SQLi defense, HTTP security headers, secret management, validated resume file uploads, SMTP email, Swagger/OpenAPI (`drf-spectacular`), and Postman automation.

Objectives: ✓ Configure a custom `User(AbstractUser)` model (`AUTH_USER_MODEL`) from Migration #1 supporting email-based login and role flags. • ✓ Implement complete Registration, Login, Password Change, Password Reset, and `SimpleJWT` Access + Refresh rotation/blacklist flows. • ✓ Enforce fine-grained Role-Based Access Control (`Candidate`, `Employer`, `Admin`) and object ownership permissions. • ✓ Defend against OWASP Top 10 risks: CORS misconfigurations, CSRF, XSS, SQL injection, insecure headers, secret leaks, and malicious file uploads. • ✓ Document every API automatically with OpenAPI 3 / Swagger UI (`drf-spectacular`) and verify flows via automated Postman collections.

15.1 Django Built-In Authentication System [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `django.contrib.auth` architecture: `User` • `AuthenticationBackend` • `authenticate()` • `Permission` • Production Edge-Case & Pitfall Guard (Django Built-In Authentication System) • Hands-On Python Implementation & Verification (Django Built-In Authentication System) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Built-In Authentication System

Purpose: Leveraging Django's timing-attack-resistant `authenticate()` and pluggable auth backends instead of rolling custom crypto.

What Topic Is Next → Next Topic 15.2: Default Django User Model

15.2 Default Django User Model [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fields on default `auth.User` (`username` • `email` • `password` • `is_active` • Production Edge-Case & Pitfall Guard (Default Django User Model) • Hands-On Python Implementation & Verification (Default Django User Model) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Default Django User Model

Purpose: Understanding `is_active` (soft-banning users immediately) vs `is_staff` (Django Admin access) vs `is_superuser`.

What Topic Is Next → Next Topic 15.3: Custom User Model (AbstractUser and AbstractBaseUser)

15.3 Custom User Model (AbstractUser and AbstractBaseUser) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Defining `class User(AbstractUser):` (`USERNAME_FIELD = "email"` • Unique case-insensitive email • `role` enum) • `AUTH_USER_MODEL = "accounts.User"` BEFORE initial `migrate` • Production Edge-Case & Pitfall Guard (Custom User Model (AbstractUser and AbstractBaseUser)) • Hands-On Python Implementation & Verification (Custom User Model (AbstractUser and AbstractBaseUser)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom User Model (AbstractUser and AbstractBaseUser...

Purpose: Enabling modern email + password sign-in with `UUID` primary key and `role` (`CANDIDATE`, `EMPLOYER`, `ADMIN`) from day one.

What Topic Is Next → Next Topic 15.4: User Registration REST API

15.4 User Registration REST API [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `POST /api/v1/auth/register/` (`write_only=True` password • `validate_password()` strength validators • Normalized lowercase email • Atomic profile creation) • Production Edge-Case & Pitfall Guard (User Registration REST API) • Hands-On Python Implementation & Verification (User Registration REST API) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for User Registration REST API

Purpose: Creating `User` + `CandidateProfile`/`EmployerProfile` atomically inside `transaction.atomic()` while rejecting weak passwords.

What Topic Is Next → Next Topic 15.5: User Login and Credential Verification API

15.5 User Login and Credential Verification API [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `POST /api/v1/auth/login/` generic `"Invalid email or password"` error (preventing account enumeration!) • Rate throttling • Inactive user rejection • Production Edge-Case & Pitfall Guard (User Login and Credential Verification API) • Hands-On Python Implementation & Verification (User Login and Credential Verification API) • User Login and Credential Verification API — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for User Login

Purpose: Verifying credentials securely without revealing whether a target email address is registered in the system.

What Topic Is Next → Next Topic 15.6: Secure Password Hashing (PBKDF2 and Argon2)

15.6 Secure Password Hashing (PBKDF2 and Argon2) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `PASSWORD_HASHERS` (`Argon2PasswordHasher` primary • `PBKDF2PasswordHasher`) • Per-user cryptographic salt • Work factors • Production Edge-Case & Pitfall Guard (Secure Password Hashing (PBKDF2 and Argon2)) • Hands-On Python Implementation & Verification (Secure Password Hashing (PBKDF2 and Argon2)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secure Password Hashing (PBKDF2 and Argon2)

Purpose: Configuring `django[argon2]` (`Argon2id`) as primary password hasher defeating GPU rainbow-table cracking.

What Topic Is Next → Next Topic 15.7: Password Reset Token Workflow

15.7 Password Reset Token Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `PasswordResetTokenGenerator` (`default_token_generator`) • `urlsafe_base64_encode(uid)` • Single-use time-bounded tokens • Revoking existing refresh tokens on reset • Production Edge-Case & Pitfall Guard (Password Reset Token Workflow) • Hands-On Python Implementation & Verification (Password Reset Token Workflow) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Password Reset Token Workflow

Purpose: Sending time-limited single-use reset links via background Celery email while always returning `200 OK` on `/forgot-password` (enumeration-safe!).

What Topic Is Next → Next Topic 15.8: Session-Based Authentication

15.8 Session-Based Authentication [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stateful `sessionid` (`HttpOnly` • `Secure` • `SameSite=Lax`) cookie + Redis session store (`SESSION_ENGINE`) + mandatory CSRF protection • Production Edge-Case & Pitfall Guard (Session-Based Authentication) • Hands-On Python Implementation & Verification (Session-Based Authentication) • Session-Based Authentication — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Session-Based Authentication

Purpose: Powering Django Admin and same-site server-rendered dashboards securely.

What Topic Is Next → Next Topic 15.9: DRF Token Authentication

15.9 DRF Token Authentication [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `rest_framework.authtoken` opaque DB-backed token vs stateless JWT trade-offs (1 DB lookup per request & single non-expiring token per user) • Production Edge-Case & Pitfall Guard (DRF Token Authentication) • Hands-On Python Implementation & Verification (DRF Token Authentication) • DRF Token Authentication — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DRF Token Authentication • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding simple CLI/service API keys vs why multi-device SPAs/mobile apps prefer short-lived JWT + Refresh rotation.

What Topic Is Next → Next Topic 15.10: JSON Web Token (JWT) Structure and Signing

15.10 JSON Web Token (JWT) Structure and Signing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `header.payload.signature` Base64URL anatomy (`alg` • `sub` • `exp` • `iat` • Production Edge-Case & Pitfall Guard (JSON Web Token (JWT) Structure and Signing) • Hands-On Python Implementation & Verification (JSON Web Token (JWT) Structure and Signing) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JSON Web Token (JWT) Structure

Purpose: Verifying user identity and role claims statelessly across API Gateway and downstream services in `< 0.2ms`.

What Topic Is Next → Next Topic 15.11: SimpleJWT Integration in DRF

15.11 SimpleJWT Integration in DRF [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `djangorestframework-simplejwt` • `JWTAuthentication` • `TokenObtainPairView` • `TokenRefreshView` • Production Edge-Case & Pitfall Guard (SimpleJWT Integration in DRF) • Hands-On Python Implementation & Verification (SimpleJWT Integration in DRF) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SimpleJWT Integration in DRF

Purpose: Wiring `/api/v1/auth/token/` and `/api/v1/auth/token/refresh/` cleanly into DRF.

What Topic Is Next → Next Topic 15.12: Short-Lived Access Token Handling

15.12 Short-Lived Access Token Handling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Setting `ACCESS_TOKEN_LIFETIME = timedelta(minutes=15)` • Skew leeway • `Bearer <token>` verification • Minimizing revocation window • Production Edge-Case & Pitfall Guard (Short-Lived Access Token Handling) • Hands-On Python Implementation & Verification (Short-Lived Access Token Handling) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Short-Lived Access Token Handling

Purpose: Limiting exposure window of any leaked access token to 15 minutes while keeping API calls 100% DB-free.

What Topic Is Next → Next Topic 15.13: Refresh Token Rotation and Blacklisting

15.13 Refresh Token Rotation and Blacklisting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ROTATE_REFRESH_TOKENS = True` • `BLACKLIST_AFTER_ROTATION = True` • `rest_framework_simplejwt.token_blacklist` • `POST /auth/logout/` (`TokenBlacklistView`) • Production Edge-Case & Pitfall Guard (Refresh Token Rotation and Blacklisting) • Hands-On Python Implementation & Verification (Refresh Token Rotation and Blacklisting) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Refresh Token Rotation

Purpose: Invalidating old refresh tokens immediately on every refresh and on explicit user logout!

What Topic Is Next → Next Topic 15.14: Authentication vs Authorization

15.14 Authentication vs Authorization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Clear architectural separation: Authentication (`request.user` identity -> `401`) vs Authorization (role + resource ownership policy -> `403`) • Production Edge-Case & Pitfall Guard (Authentication vs Authorization) • Hands-On Python Implementation & Verification (Authentication vs Authorization) • Authentication vs Authorization — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Authentication vs Authorization • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Separating JWT identity parsing cleanly from declarative permission classes.

What Topic Is Next → Next Topic 15.15: Role-Based Access Control (RBAC) Permissions

15.15 Role-Based Access Control (RBAC) Permissions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `IsCandidate` • `IsEmployer` • `IsPlatformAdmin` DRF permission classes + object-level ownership (`application.job.company.owner_id == request.user.id`) • Production Edge-Case & Pitfall Guard (Role-Based Access Control (RBAC) Permissions) • Hands-On Python Implementation & Verification (Role-Based

Access Control (RBAC) Permissions) • Role-Based Access Control (RBAC) Permissions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Role-Based Access Control (RBAC) Permissions

Purpose: Ensuring ONLY verified Employers can post jobs, ONLY Candidates can apply, and Employers ONLY view applications submitted to THEIR own jobs!

What Topic Is Next → Next Topic 15.16: Django Groups and Object-Level Permissions

15.16 Django Groups and Object-Level Permissions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Group` • `Permission` (`jobs.publish_jobposting`) • `user.has_perm()` • Seeding standard staff roles (`Moderator` • Production Edge-Case & Pitfall Guard (Django Groups and Object-Level Permissions) • Hands-On Python Implementation & Verification (Django Groups and Object-Level Permissions) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Groups

Purpose: Granting content moderators permission to review flagged job posts without giving full `is_superuser` database power.

What Topic Is Next → Next Topic 15.17: Cross-Origin Resource Sharing (django-cors-headers)

15.17 Cross-Origin Resource Sharing (django-cors-headers) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Same-Origin Policy • `OPTIONS` preflight requests • `django-cors-headers` middleware placement (top of `MIDDLEWARE`!) • Explicit `CORS_ALLOWED_ORIGINS` allowlist • Production Edge-Case & Pitfall Guard (Cross-Origin Resource Sharing (django-cors-headers)) • Hands-On Python Implementation & Verification (Cross-Origin Resource Sharing (django-cors-headers)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Origin Resource Sharing (django-cors-headers)

Purpose: Allowing `https://app.mockattempt.com` React SPA to call `https://api.mockattempt.com` securely.

What Topic Is Next → Next Topic 15.18: Cross-Site Request Forgery (CSRF) Protection

15.18 Cross-Site Request Forgery (CSRF) Protection [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How browsers auto-attach cookies cross-site • `CsrfViewMiddleware` • `SameSite=Lax/Strict` • Why `Authorization: Bearer` header APIs are immune whereas cookie auth MUST enforce CSRF • Production Edge-Case & Pitfall Guard (Cross-Site Request Forgery (CSRF) Protection) • Hands-On Python Implementation & Verification (Cross-Site Request Forgery (CSRF) Protection) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Site Request Forgery (CSRF) Protection

Purpose: Keeping CSRF strictly enabled on cookie-authenticated endpoints while understanding stateless Bearer header immunity.

What Topic Is Next → Next Topic 15.19: Cross-Site Scripting (XSS) Prevention

15.19 Cross-Site Scripting (XSS) Prevention [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stored vs Reflected XSS • HTML escaping • Sanitizing rich-text Markdown/HTML descriptions (`nh3` / `bleach`) • `Content-Type: application/json` (`nosniff`) • Production Edge-Case & Pitfall Guard (Cross-Site Scripting (XSS) Prevention) • Hands-On Python Implementation & Verification (Cross-Site Scripting (XSS) Prevention) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cross-Site Scripting (XSS) Prevention

Purpose: Sanitizing employer job descriptions on input/render so `<script>` or `<img onerror=...>` payloads are stripped cleanly.

What Topic Is Next → Next Topic 15.20: SQL Injection Prevention in Django ORM

15.20 SQL Injection Prevention in Django ORM [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parameterized Django `QuerySet` safety vs dangerous escape hatches (`raw()` • `extra()` • `RawSQL`) when developers interpolate f-strings • Production Edge-Case & Pitfall Guard (SQL Injection Prevention in Django ORM) • Hands-On Python Implementation & Verification (SQL Injection Prevention in Django ORM) • SQL Injection Prevention in Django ORM — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQL Injection Prevention in Django ORM

Purpose: Keeping 100% of dynamic filters inside parameterized `QuerySet` / `Q()` expressions (or `params=[...]` if `raw()` is ever needed).

What Topic Is Next → Next Topic 15.21: HTTP Security Headers (HSTS, CSP, X-Frame-Options)

15.21 HTTP Security Headers (HSTS, CSP, X-Frame-Options) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SecurityMiddleware`: `SECURE_SSL_REDIRECT` • `SECURE_HSTS_SECONDS = 31536000` • `SECURE_CONTENT_TYPE_NOSNIFF = True` • `X_FRAME_OPTIONS = "DENY"` • Production Edge-Case & Pitfall Guard (HTTP Security Headers (HSTS, CSP, X-Frame-Options)) • Hands-On Python Implementation & Verification (HTTP Security Headers (HSTS, CSP, X-Frame-Options)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP Security Headers (HSTS, CSP, X-Frame-Options)

Purpose: Passing `python manage.py check --deploy` with zero security warnings before every production release!

What Topic Is Next → Next Topic 15.22: Environment Variables and Secret Management

15.22 Environment Variables and Secret Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zero secrets in Git • Rotating leaked keys immediately • `django-environ` / `pydantic-settings` • AWS Secrets Manager / GitHub Encrypted Secrets • Production Edge-Case & Pitfall Guard (Environment Variables and Secret Management) • Hands-On Python Implementation & Verification (Environment Variables and Secret Management) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment Variables

Purpose: Loading `DJANGO_SECRET_KEY`, `DATABASE_URL`, `REDIS_URL`, and `AWS_SECRET_ACCESS_KEY` exclusively from runtime environment.

What Topic Is Next → Next Topic 15.23: Multipart File and Resume Upload APIs

15.23 Multipart File and Resume Upload APIs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `MultiPartParser` & `FormParser` • `FileField(upload_to=...)` with UUID randomized filenames (preventing path traversal & overwrites!) • S3 `django-storages` • Production Edge-Case & Pitfall Guard (Multipart File and Resume Upload APIs) • Hands-On Python Implementation & Verification (Multipart File and Resume Upload APIs) • Multipart File and Resume Upload APIs — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular

architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multipart File

Purpose: Accepting candidate PDF resume uploads via `POST /api/v1/candidates/me/resumes/` stored under `resumes/<uuid4>.pdf`.

What Topic Is Next → Next Topic 15.24: Upload File Size, MIME and Extension Validation

15.24 Upload File Size, MIME and Extension Validation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 3-layer upload validation: max size (`<= 5 MB`) • Extension allowlist (`.pdf` • `.docx`) • Magic-byte header inspection (`%PDF-` / ZIP PK header — never trust client `content_type` header alone!) • Production Edge-Case & Pitfall Guard (Upload File Size, MIME and Extension Validation) • Hands-On Python Implementation & Verification (Upload File Size, MIME and Extension Validation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Upload File Size

Purpose: Rejecting a malicious `malware.exe` renamed to `resume.pdf` by verifying the first 5 magic bytes equal `b"%PDF-"`!

What Topic Is Next → Next Topic 15.25: Transactional Email Integration via SMTP

15.25 Transactional Email Integration via SMTP [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `EMAIL_BACKEND` (`console` in dev • `locmem` in tests • `smtp`/SES in prod) • `EmailMultiAlternatives` (plain text + HTML) • Production Edge-Case & Pitfall Guard (Transactional Email Integration via SMTP) • Hands-On Python Implementation & Verification (Transactional Email Integration via SMTP) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Transactional Email Integration via SMTP

Purpose: Sending registration verification, password reset, and application status update emails cleanly.

What Topic Is Next → Next Topic 15.26: Swagger Interactive API Documentation

15.26 Swagger Interactive API Documentation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Serving interactive `/api/schema/swagger-ui/` • `/api/schema/redoc/` with Bearer JWT `Authorize` button for frontend & QA collaboration • Production Edge-Case & Pitfall Guard (Swagger Interactive API Documentation) • Hands-On Python Implementation & Verification (Swagger Interactive API Documentation) • Swagger Interactive API Documentation — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Swagger Interactive API Documentation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Giving React/Angular developers live "Try it out" API documentation generated directly from DRF serializers.

What Topic Is Next → Next Topic 15.27: OpenAPI 3 Schema Generation (drf-spectacular)

15.27 OpenAPI 3 Schema Generation (drf-spectacular) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `drf-spectacular` • `AutoSchema` • `@extend_schema(summary=... • Request=... • Production Edge-Case & Pitfall Guard (OpenAPI 3 Schema Generation (drf-spectacular)) • Hands-On Python Implementation & Verification (OpenAPI 3 Schema Generation (drf-spectacular)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OpenAPI 3 Schema Generation (drf-spectacular)

Purpose: Annotating custom `APIView`s and `@action` endpoints with `@extend_schema` so generated OpenAPI 3 schemas have zero warnings.

What Topic Is Next → Next Topic 15.28: API Testing and Environments with Postman

15.28 API Testing and Environments with Postman [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Organizing Postman Collections (`Auth` • `Companies` • `Jobs` • `Applications`) • Production Edge-Case & Pitfall Guard (API Testing and Environments with Postman) • Hands-On Python Implementation & Verification (API Testing and Environments with Postman) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Testing

Purpose: Exporting a turnkey `postman_collection.json` so reviewers and teammates can run end-to-end Register -> Login -> Post Job -> Apply flows in 1 click.

What Topic Is Next → Module 15 Capstone (Week 15 Lab Build: Argon2 + SimpleJWT RBAC, PDF Resume Upload & OpenAPI Suite) → Next Topic 16.1: Scalable Django Project Structure

Module 15 Hands-On Capstone Project (100 Marks): Week 15 Lab Build: Argon2 + SimpleJWT RBAC, PDF Resume Upload & OpenAPI Suite — Harden your DRF API with custom email `User`, Argon2id hashing, rotated/blacklisted `SimpleJWT` tokens, Candidate/Employer RBAC, `%PDF-` magic-byte validated resume uploads, OWASP security h...

MODULE 16 OF 24 • STAGE 4 • 25 LIVE + LAB HOURS • 21 TOPICS (168 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 16: Django Production Development and Backend Architecture

Architect maintainable, scalable production Django backends and deliver Major Project 4: scalable app separation, Service & Selector/Repository layers, custom middleware, signals vs explicit service calls, custom managers/QuerySets, eliminating N+1 queries, `transaction.atomic()` + `on_commit()`, Django cache + Redis backend, Celery + task queues + background email jobs, structured logging, Django TestCase + DRF `APIClient`, split dev/prod settings, WSGI vs ASGI, production prep, and Major Project 4 (`Job Portal Backend using Django REST Framework`).

Objectives: ✓ Structure multi-app Django codebases cleanly using thin Views, rich `services.py` write commands, and optimized `selectors.py` read queries. • ✓ Coordinate multi-model writes safely inside `transaction.atomic()` while scheduling Celery tasks strictly inside `transaction.on_commit()`! • ✓ Accelerate hot read endpoints with `django-redis` caching and offload slow emails/notifications to Celery workers. • ✓ Write comprehensive automated API test suites with DRF `APIClient`, fixtures/factories, and `assertNumQueries` N+1 guards. • ✓ Deliver Major Project 4: a complete, tested, documented Job Portal Backend using Django REST Framework & PostgreSQL.

16.1 Scalable Django Project Structure [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Organizing `config/settings/{base • Dev • Test • Prod}.py` • Production Edge-Case & Pitfall Guard (Scalable Django Project Structure) • Hands-On Python Implementation & Verification (Scalable Django Project Structure) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Scalable Django Project

Structure

Purpose: Scaling a Django codebase cleanly across 10+ models and 50+ REST endpoints without circular-import tangles.

What Topic Is Next → Next Topic 16.2: Feature-Based Application Separation

16.2 Feature-Based Application Separation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Bounded domain cohesion per app • Unidirectional dependency rules (`applications` -> `jobs` -> `companies` -> `accounts`) • Avoiding circular FK/service imports • Production Edge-Case & Pitfall Guard (Feature-Based Application Separation) • Hands-On Python Implementation & Verification (Feature-Based Application Separation) • Feature-Based Application Separation — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Feature-Based Application Separation

Purpose: Referencing `"companies.Company"` by string in `ForeignKey` to eliminate import-time circular dependencies.

What Topic Is Next → Next Topic 16.3: Service Layer Pattern in Django

16.3 Service Layer Pattern in Django [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Extracting business write rules (`submit_job_application(* • Candidate • Job • Resume_file)`) into `services.py` keeping DRF ViewSets thin HTTP adapters • Production Edge-Case & Pitfall Guard (Service Layer Pattern in Django) • Hands-On Python Implementation & Verification (Service Layer Pattern in Django) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Layer Pattern in Django

Purpose: Reusing identical `submit_job_application` validation, DB transaction, and Celery trigger logic from REST API, CLI command, and tests!

What Topic Is Next → Next Topic 16.4: Repository / Selector Data Access Abstraction

16.4 Repository / Selector Data Access Abstraction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Centralizing optimized read queries (`list_published_jobs` • `get_employer_job_detail`) in `selectors.py` with baked-in `select_related`/`prefetch_related` • Production Edge-Case & Pitfall Guard (Repository / Selector Data Access Abstraction) • Hands-On Python Implementation & Verification (Repository / Selector Data Access Abstraction) • Repository / Selector Data Access Abstraction — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Repository / Selector Data Access Abstraction • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Guaranteeing every caller reading job listings automatically gets N+1-free eager loading and active-tenant filtering.

What Topic Is Next → Next Topic 16.5: Custom Request-Response Middleware

16.5 Custom Request-Response Middleware [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Writing sync/async-compatible middleware (`__init__(get_response)` + `__call__(request)`) • Injecting `X-Request-ID` & measuring `X-Response-Time-Ms` • Production Edge-Case & Pitfall Guard (Custom Request-Response Middleware) • Hands-On Python Implementation & Verification (Custom Request-Response Middleware) • Custom Request-Response Middleware — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Request-Response Middleware • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Attaching a unique UUID `request.request_id` to every incoming request and response header for end-to-end log tracing.

What Topic Is Next → Next Topic 16.6: Django Signals and Decoupled Hooks

16.6 Django Signals and Decoupled Hooks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `post_save` • `pre_delete` • `@receiver` • Why explicit `services.py` calls are strongly preferred over hidden implicit `post_save` signal chains! • Production Edge-Case & Pitfall Guard (Django Signals and Decoupled Hooks) • Hands-On Python Implementation & Verification (Django Signals and Decoupled Hooks) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Signals

Purpose: Knowing how signals work (e.g., cache invalidation) while avoiding "spaghetti signals" where `.save()` mysteriously triggers 5 hidden emails!

What Topic Is Next → Next Topic 16.7: Custom Model Managers and QuerySets

16.7 Custom Model Managers and QuerySets [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Writing `class JobQuerySet(models.QuerySet): def published(self): ... def with_relations(self): ...` exposed via `objects = JobQuerySet.as_manager()` • Production Edge-Case & Pitfall Guard (Custom Model Managers and QuerySets) • Hands-On Python Implementation & Verification (Custom Model Managers and QuerySets) • Custom Model Managers and QuerySets — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Model Managers • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Writing readable, chainable domain queries: `JobPosting.objects.published().with_relations().in_city("Pune")`.

What Topic Is Next → Next Topic 16.8: Eliminating N+1 Queries in DRF Serializers

16.8 Eliminating N+1 Queries in DRF Serializers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Spotting `SerializerMethodField` per-row queries (`obj.applications.count()` firing `N` `SELECT COUNT(*)` queries!) • Fixing with QuerySet `annotate(applications_count=Count("applications"))` • Production Edge-Case & Pitfall Guard (Eliminating N+1 Queries in DRF Serializers) • Hands-On Python Implementation & Verification (Eliminating N+1 Queries in DRF Serializers) • Eliminating N+1 Queries in DRF Serializers — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Eliminating N+1 Queries in DRF Serializers • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Precomputing `applications_count` in SQL `annotate()` once instead of running 20 extra `COUNT(*)` queries per page!

What Topic Is Next → Next Topic 16.9: Database Atomicity with transaction.atomic

16.9 Database Atomicity with transaction.atomic [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@transaction.atomic` / `with transaction.atomic():` • Nested savepoints • `select_for_update()` row locks • `transaction.on_commit()` for Celery/email side effects • Production Edge-Case & Pitfall Guard (Database Atomicity with transaction.atomic) • Hands-On Python Implementation & Verification (Database Atomicity with transaction.atomic) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database Atomicity with transaction.atomic

Purpose: Locking limited interview slots with `select_for_update()` and queuing Celery notification tasks ONLY inside `transaction.on_commit()`!

What Topic Is Next → Next Topic 16.10: Django Cache Framework

16.10 Django Cache Framework [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `CACHES` configuration • Low-level `cache.get(key)` / `cache.set(key • Val • Timeout=300)` / `cache.get_or_set()` • Production Edge-Case & Pitfall Guard (Django Cache Framework) • Hands-On Python Implementation & Verification (Django Cache Framework) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Cache Framework

Purpose: Caching public homepage job category counts and featured jobs for 5 minutes with explicit invalidation on publish.

What Topic Is Next → Next Topic 16.11: Redis Cache Backend Integration

16.11 Redis Cache Backend Integration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `django.core.cache.backends.redis.RedisCache` (`REDIS_URL`) • TTL tuning • Atomic `cache.incr()` • Graceful degradation when Redis restarts • Production Edge-Case & Pitfall Guard (Redis Cache Backend Integration) • Hands-On Python Implementation & Verification (Redis Cache Backend Integration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis Cache Backend Integration

Purpose: Sharing cache entries and DRF rate-throttle counters consistently across all Gunicorn worker processes via Redis 7.

What Topic Is Next → Next Topic 16.12: Background Jobs with Celery Fundamentals

16.12 Background Jobs with Celery Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Installing `celery[redis]` • Configuring `config/celery.py` • `@shared_task` • `CELERY_TASK_SERIALIZER = "json"` (never pickle!) • Production Edge-Case & Pitfall Guard (Background Jobs with Celery Fundamentals) • Hands-On Python Implementation & Verification (Background Jobs with Celery Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Background Jobs with Celery Fundamentals

Purpose: Returning `201 Created` in `18ms` on job application while a Celery worker sends confirmation emails asynchronously.

What Topic Is Next → Next Topic 16.13: Asynchronous Task Queues in Django

16.13 Asynchronous Task Queues in Django [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Starting `celery -A config worker -l info` • Idempotent task design • `autoretry_for=(TimeoutError • `retry_backoff=True` • Production Edge-Case & Pitfall Guard (Asynchronous Task Queues in Django) • Hands-On Python Implementation & Verification (Asynchronous Task Queues in Django) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Asynchronous Task Queues in Django

Purpose: Ensuring retried Celery tasks never send duplicate emails by checking/recording a notification state column first.

What Topic Is Next → Next Topic 16.14: Background Transactional Email Processing

16.14 Background Transactional Email Processing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Offloading registration welcome • Password reset • Application-shortlisted emails to Celery via `transaction.on_commit()` with delivery status tracking • Production Edge-Case & Pitfall Guard (Background Transactional Email Processing) • Hands-On Python Implementation & Verification (Background Transactional Email Processing) • Background Transactional Email Processing — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Background Transactional Email Processing

Purpose: Keeping API p99 latency `< 40ms` even when external SMTP providers take 1,200ms per email.

What Topic Is Next → Next Topic 16.15: Structured Production Logging

16.15 Structured Production Logging [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring Django `LOGGING` dict (`version: 1` • JSON/structured formatters • `request_id` filter • `django.request` & `apps` loggers • Production Edge-Case & Pitfall Guard (Structured Production Logging) • Hands-On Python Implementation & Verification (Structured Production Logging) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Structured Production Logging

Purpose: Streaming structured JSON logs to stdout (`StreamHandler`) in Docker so CloudWatch/ELK indexes `level`, `request_id`, and `user_id`.

What Topic Is Next → Next Topic 16.16: Django Automated Test Framework

16.16 Django Automated Test Framework [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `django.test.TestCase` (per-test DB transaction rollback!) vs `SimpleTestCase` (no DB) • `setUpTestData` class-level fixture speedup • `override_settings` • Production Edge-Case & Pitfall Guard (Django Automated Test Framework) • Hands-On Python Implementation & Verification (Django Automated Test Framework) • Django Automated Test Framework — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Automated Test Framework

Purpose: Using `@classmethod def setUpTestData(cls):` to seed read-mostly reference data once per test class for 5x faster test suites.

What Topic Is Next → Next Topic 16.17: REST Endpoint Testing with DRF APIClient

16.17 REST Endpoint Testing with DRF APIClient [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `rest_framework.test.APITestCase` & `APIClient` • `client.force_authenticate(user=...)` vs real JWT header tests • Testing `200/201/204/400/401/403/404/409` • Production Edge-Case & Pitfall Guard (REST Endpoint Testing with DRF APIClient) • Hands-On Python Implementation & Verification (REST Endpoint Testing with DRF APIClient) • REST Endpoint Testing with DRF APIClient — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production

Runtime Execution & Memory Lifecycle for REST Endpoint Testing with DRF APIClient

Purpose: Testing complete Employer vs Candidate RBAC boundaries: proving Employer B gets `403` when trying to edit Employer A's job!

What Topic Is Next → Next Topic 16.18: Isolated Test Database Lifecycle

16.18 Isolated Test Database Lifecycle [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How Django creates/destroys `test_<dbname>` • `--keepdb` local iteration speedup • `--parallel` • Testing real PostgreSQL constraints (`CheckConstraint` • Production Edge-Case & Pitfall Guard (Isolated Test Database Lifecycle) • Hands-On Python Implementation & Verification (Isolated Test Database Lifecycle) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Isolated Test Database Lifecycle

Purpose: Running tests against real PostgreSQL in CI so PostgreSQL-specific indexes, `DISTINCT ON`, and constraints are verified.

What Topic Is Next → Next Topic 16.19: Split Development, Staging and Production Settings

16.19 Split Development, Staging and Production Settings [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `config/settings/base.py` • `dev.py` • `test.py` • `prod.py` selected via `DJANGO_SETTINGS_MODULE` • Production Edge-Case & Pitfall Guard (Split Development, Staging and Production Settings) • Hands-On Python Implementation & Verification (Split Development, Staging and Production Settings) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Split Development

Purpose: Enforcing HTTPS, HSTS, Redis cache, JSON renderer, and connection pooling automatically whenever `DJANGO_SETTINGS_MODULE=config.settings.prod`.

What Topic Is Next → Next Topic 16.20: WSGI vs ASGI Application Server Interfaces

16.20 WSGI vs ASGI Application Server Interfaces [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Synchronous PEP 3333 WSGI (`config.wsgi:application` with Gunicorn sync/gthread workers) vs asynchronous ASGI 3.0 (`config.asgi:application` with Uvicorn workers for async views/WebSockets) • Production Edge-Case & Pitfall Guard (WSGI vs ASGI Application Server Interfaces) • Hands-On Python Implementation & Verification (WSGI vs ASGI Application Server Interfaces) • WSGI vs ASGI Application Server Interfaces — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for WSGI vs ASGI Application Server Interfaces • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Understanding why classic synchronous Django ORM apps shine on `gunicorn config.wsgi` while async views & FastAPI run on ASGI `uvicorn`!

What Topic Is Next → Next Topic 16.21: Django Production Deployment Checklist

16.21 Django Production Deployment Checklist [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pre-launch verification: `DEBUG=False` • Rotated `SECRET_KEY` • `ALLOWED_HOSTS` • `CSRF_TRUSTED_ORIGINS` • Production Edge-Case & Pitfall Guard (Django Production Deployment Checklist) • Hands-On Python Implementation & Verification (Django Production Deployment Checklist) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Production Deployment Checklist

Purpose: Delivering Major Project 4 (`Job Portal Backend using Django REST Framework`) production-ready with PostgreSQL, JWT, RBAC, Resume Upload, Swagger, and Automated Tests!

What Topic Is Next → Module 16 Capstone (Project 4: Job Portal Backend using Django REST Framework) → Next Topic 17.1: FastAPI Architecture, Starlette and ASGI Use Cases

Module 16 Hands-On Capstone Project (100 Marks): Project 4: Job Portal Backend using Django REST Framework — Build a complete, secure, database-driven Job Portal REST API using Django 5, Django REST Framework, PostgreSQL 16, SimpleJWT, Role-Based Authorization, validated PDF Resume Uploads, asynchr...

MODULE 17 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 25 TOPICS (200 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 17: FastAPI Fundamentals and API Development

Master modern high-performance Python API engineering with FastAPI & Pydantic v2: Starlette + Pydantic architecture, Uvicorn ASGI setup, project structure, path operations (`GET`/`POST`/`PUT`/`PATCH`/`DELETE`), typed path/query/body parameters, Pydantic v2 `BaseModel`, `Field()` constraints, `@field_validator`/`@model_validator`, `response_model` output filtering, status codes, `Depends()` dependency injection, reusable services, headers/cookies, `UploadFile`, `BackgroundTasks`, `HTTPException` & custom handlers, middleware, CORS, OpenAPI/Swagger UI/ReDoc, and environment settings.

Objectives: ✓ Explain FastAPI's Starlette (ASGI) + Pydantic v2 (Rust `pydantic-core`) architecture and boot apps cleanly with Uvicorn. • ✓ Define self-validating, self-documenting endpoints using typed Path, Query (`Annotated[..., Query(...)]`), Header, Cookie, and JSON Body parameters. • ✓ Author strict Pydantic v2 input/output schemas (`UserCreate` vs `UserOut` via `response_model`) preventing accidental secret leakage. • ✓ Leverage FastAPI's `Depends()` Dependency Injection system for reusable pagination, auth, and service dependencies (and `dependency_overrides` in tests!). • ✓ Handle `UploadFile` streams, lightweight `BackgroundTasks`, custom exception handlers, ASGI middleware, CORS, and `/docs` + `/redoc`.

17.1 FastAPI Architecture, Starlette and ASGI Use Cases [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: How FastAPI combines Starlette (ASGI routing/WebSockets/middleware) + Pydantic v2 (Rust `pydantic-core` validation) + automatic OpenAPI 3.1 generation • Production Edge-Case & Pitfall Guard (FastAPI Architecture, Starlette and ASGI Use Cases) • Hands-On Python Implementation & Verification (FastAPI Architecture, Starlette and ASGI Use Cases) • FastAPI Architecture, Starlette and ASGI Use Cases — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for FastAPI Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Building low-latency async microservices, ML/AI inference gateways, and high-concurrency I/O APIs.

What Topic Is Next → Next Topic 17.2: FastAPI and Uvicorn Environment Setup

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

Subtopics Covered: Installing `fastapi` + `uvicorn[standard]` (`uvloop` + `httptools`) • Running `uvicorn app.main:app --reload` in dev vs `--no-reload` in prod • Production Edge-Case & Pitfall Guard (FastAPI and Uvicorn Environment Setup) • Hands-On Python Implementation & Verification (FastAPI and Uvicorn Environment Setup) • FastAPI and Uvicorn Environment Setup — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for FastAPI • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running live-reloading local ASGI development on port `8000` with `uvicorn[standard]`.

What Topic Is Next → Next Topic 17.3: Scalable FastAPI Project Organization

17.3 Scalable FastAPI Project Organization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Domain/layered layout: `app/{main.py • Core/ • Api/v1/routers/ • Schemas/ • Production Edge-Case & Pitfall Guard (Scalable FastAPI Project Organization) • Hands-On Python Implementation & Verification (Scalable FastAPI Project Organization) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Scalable FastAPI Project Organization

Purpose: Separating Pydantic API `schemas/` cleanly from SQLAlchemy DB `models/` and business `services/`.

What Topic Is Next → Next Topic 17.4: Path Operations (GET, POST, PUT, PATCH, DELETE)

17.4 Path Operations (GET, POST, PUT, PATCH, DELETE) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@router.get()` • `.post()` • `.put()` • `.patch()` • Production Edge-Case & Pitfall Guard (Path Operations (GET, POST, PUT, PATCH, DELETE)) • Hands-On Python Implementation & Verification (Path Operations (GET, POST, PUT, PATCH, DELETE)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Path Operations (GET, POST, PUT, PATCH, DELETE)

Purpose: Registering static `/users/me` above dynamic `/users/{user_id}` so `"me"` isn't parsed as an integer `user_id` (`422`)!

What Topic Is Next → Next Topic 17.5: Typed Path Parameters

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

Subtopics Covered: Automatic conversion & `422` validation for `int` • `UUID` • `StrEnum` path parameters • Plus `Annotated[int • Production Edge-Case & Pitfall Guard (Typed Path Parameters) • Hands-On Python Implementation & Verification (Typed Path Parameters) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed Path Parameters

Purpose: Validating `product_id: Annotated[int, Path(ge=1)]` and `status: OrderStatus` (`StrEnum`) before handler code even runs!

What Topic Is Next → Next Topic 17.6: Query Parameters and Filtering

17.6 Query Parameters and Filtering [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Optional defaults (`q: str | None = None`) • Required queries • `Annotated[int • Query(ge=1 • Production Edge-Case & Pitfall Guard (Query Parameters and Filtering) • Hands-On Python Implementation & Verification (Query Parameters and Filtering) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Query Parameters

Purpose: Bounding `limit: Annotated[int, Query(ge=1, le=100)] = 20` and `offset: Annotated[int, Query(ge=0)] = 0` automatically.

What Topic Is Next → Next Topic 17.7: JSON Request Bodies

17.7 JSON Request Bodies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automatic JSON parsing into Pydantic `BaseModel` parameters • Combining Path + Query + Body parameters cleanly • `Body(embed=True)` • Production Edge-Case & Pitfall Guard (JSON Request Bodies) • Hands-On Python Implementation & Verification (JSON Request Bodies) • JSON Request Bodies — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for JSON Request Bodies

Purpose: Accepting `product_id` in path + validated `ProductUpdate` JSON payload in `PATCH /api/v1/products/{product_id}`.

What Topic Is Next → Next Topic 17.8: Pydantic v2 Data Validation Engine

17.8 Pydantic v2 Data Validation Engine [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pydantic v2 Rust `pydantic-core` speed • `model_validate()` • `model_dump(exclude_unset=True)` • `ValidationError` structured JSON details • Production Edge-Case & Pitfall Guard (Pydantic v2 Data Validation Engine) • Hands-On Python Implementation & Verification (Pydantic v2 Data Validation Engine) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pydantic v2 Data Validation Engine

Purpose: Setting `model_config = ConfigDict(extra="forbid", str_strip_whitespace=True)` on write schemas to catch misspelled client keys immediately!

What Topic Is Next → Next Topic 17.9: Defining Schemas with BaseModel

17.9 Defining Schemas with BaseModel [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Separating `ProductBase` • `ProductCreate` • `ProductPatch` (all fields optional `| None = None`) • `ProductOut` (`from_attributes=True`) • Production Edge-Case & Pitfall Guard (Defining Schemas with BaseModel) • Hands-On Python Implementation & Verification (Defining Schemas with BaseModel) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Defining Schemas with BaseModel

Purpose: Ensuring `UserCreate` accepts `password: str` while `UserOut` physically does NOT have a `password` or `hashed_password` field!

What Topic Is Next → Next Topic 17.10: Field Constraints with Pydantic Field()

17.10 Field Constraints with Pydantic Field() [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Field(min_length=2 • Max_length=120 • Pattern=...)` • `gt=0` • Production Edge-Case & Pitfall Guard (Field Constraints with Pydantic Field()) • Hands-On Python Implementation & Verification (Field Constraints with Pydantic Field()) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Field Constraints with Pydantic Field()

Purpose: Validating `sku: str = Field(pattern=r"^SKU-[A-Z0-9]{4,12}$")`, `price: Decimal = Field(gt=0)`, and `stock: int = Field(ge=0)` declaratively.

What Topic Is Next → Next Topic 17.11: Custom Validators (@field_validator and @model_validator)

17.11 Custom Validators (@field_validator and @model_validator) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pydantic v2 `@field_validator("email" • Mode="before")` (`@classmethod`) normalization • `@model_validator(mode="after")` cross-field invariants • Production Edge-Case & Pitfall Guard (Custom Validators (@field_validator and @model_validator)) • Hands-On Python Implementation & Verification (Custom Validators (@field_validator and @model_validator)) • Custom Validators (@field_validator and @model_validator) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom Validators (@field_validator and @model_valid...

Purpose: Normalizing emails lowercase via `@field_validator` and verifying `discount_price <= mrp_price` via `@model_validator(mode="after")`.

What Topic Is Next → Next Topic 17.12: Response Models and Output Filtering (response_model)

17.12 Response Models and Output Filtering (response_model) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `response_model=ProductOut` (or `-> ProductOut` return annotation) • `ConfigDict(from_attributes=True)` ORM serialization • `response_model_exclude_none` • Production Edge-Case & Pitfall Guard (Response Models and Output Filtering (response_model)) • Hands-On Python Implementation & Verification (Response Models and Output Filtering (response_model)) • Response Models and Output Filtering (response_model) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Response Models

Purpose: Returning a SQLAlchemy `User` entity directly from a route while `response_model=UserOut` strips `hashed_password` and internal flags automatically!

What Topic Is Next → Next Topic 17.13: Explicit HTTP Status Codes

17.13 Explicit HTTP Status Codes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `status_code=status.HTTP_201_CREATED` on `POST` • `status.HTTP_204_NO_CONTENT` (returning `None`) on `DELETE` • `status.HTTP_202_ACCEPTED` for queued async jobs • Production Edge-Case & Pitfall Guard (Explicit HTTP Status Codes) • Hands-On Python Implementation & Verification (Explicit HTTP Status Codes) • Explicit HTTP Status Codes — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Explicit HTTP Status Codes

Purpose: Returning `201 Created` on resource creation and `202 Accepted` when queuing an asynchronous order/report workflow.

What Topic Is Next → Next Topic 17.14: Dependency Injection with Depends()

17.14 Dependency Injection with Depends() [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from fastapi import Depends` • `Annotated[PaginationParams • Depends()]` • Sub-dependencies • Production Edge-Case & Pitfall Guard (Dependency Injection with Depends()) • Hands-On Python Implementation & Verification (Dependency Injection with Depends()) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dependency Injection with Depends()

Purpose: Injecting DB sessions, current authenticated user, and pagination cleanly — and swapping mocks in `pytest` via `app.dependency_overrides[get_db] = ...`!

What Topic Is Next → Next Topic 17.15: Reusable Service and Auth Dependencies

17.15 Reusable Service and Auth Dependencies [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Chaining `get_db -> get_product_repo -> get_product_service` • Router-level `dependencies=[Depends(require_auth)]` • Parameterized `Depends(RequireRole("admin"))` callables • Production Edge-Case & Pitfall Guard (Reusable Service and Auth Dependencies) • Hands-On Python Implementation & Verification (Reusable Service and Auth Dependencies) • Reusable Service and Auth Dependencies — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Reusable Service

Purpose: Protecting an entire `/api/v1/admin` `APIRouter` with `dependencies=[Depends(require_role("admin"))]` in one line!

What Topic Is Next → Next Topic 17.16: Header and Cookie Parameters

17.16 Header and Cookie Parameters [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Annotated[str | None • Header()] = None` (auto hyphen-to-underscore `x_idempotency_key` <- `X-Idempotency-Key`) • `Cookie()` • Setting `HttpOnly` cookies on `Response` • Production Edge-Case & Pitfall Guard (Header and Cookie Parameters) • Hands-On Python Implementation & Verification (Header and Cookie Parameters) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Header

Purpose: Extracting `Idempotency-Key: Annotated[str, Header(min_length=8)]` on `POST /orders` to prevent duplicate charges on network retries!

What Topic Is Next → Next Topic 17.17: Asynchronous File Uploads (UploadFile)

17.17 Asynchronous File Uploads (UploadFile) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `file: UploadFile = File(...)` (`SpooledTemporaryFile` spilling to disk) • Streaming chunk validation (`await file.read(65536)`) • Magic-byte + size limits • `await file.close()` • Production Edge-Case & Pitfall Guard (Asynchronous File Uploads (UploadFile)) • Hands-On Python Implementation & Verification (Asynchronous File Uploads (UploadFile)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Asynchronous File Uploads (UploadFile)

Purpose: Streaming resume/catalog CSV uploads in `64 KB` chunks rejecting files `> 5 MB` mid-stream without buffering 500 MB into RAM!

What Topic Is Next → Next Topic 17.18: In-Process Background Tasks (BackgroundTasks)

17.18 In-Process Background Tasks (BackgroundTasks) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `background_tasks.add_task(fn • *args)` running AFTER HTTP response is sent vs when durable Celery + Redis/RabbitMQ is required instead • Production Edge-Case & Pitfall Guard (In-Process Background Tasks (BackgroundTasks)) • Hands-On Python Implementation & Verification (In-Process Background Tasks (BackgroundTasks)) • In-Process Background Tasks (BackgroundTasks) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for In-Process Background Tasks (BackgroundTasks) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Using `BackgroundTasks` for lightweight best-effort audit/cache warmups while using Celery for critical retryable emails/invoices (survives container restarts!).

What Topic Is Next → Next Topic 17.19: HTTPException and Custom Exception Handlers

17.19 HTTPException and Custom Exception Handlers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `raise HTTPException(status_code=... • Detail=...)` • `@app.exception_handler(DomainError)` translating pure domain exceptions into clean HTTP responses • Overriding `RequestValidationError` • Production Edge-Case & Pitfall Guard (HTTPException and Custom Exception Handlers) • Hands-On Python Implementation & Verification (HTTPException and Custom Exception Handlers) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTPException

Purpose: Keeping `services/` 100% pure Python (raising `OutOfStockError`, `DuplicateSkuError`) and mapping them centrally in `main.py` exception handlers!

What Topic Is Next → Next Topic 17.20: Custom ASGI / HTTP Middleware

17.20 Custom ASGI / HTTP Middleware [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `@app.middleware("http")` (`call_next(request)`) • Adding `X-Request-ID` & `X-Process-Time-Ms` headers • Pure ASGI middleware ordering (LIFO stack!) • Production Edge-Case & Pitfall Guard (Custom ASGI / HTTP Middleware) • Hands-On Python Implementation & Verification (Custom ASGI / HTTP Middleware) • Custom ASGI / HTTP Middleware — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Custom ASGI / HTTP Middleware

Purpose: Measuring exact per-endpoint latency and propagating `X-Correlation-ID` across `contextvars` into every log line.

What Topic Is Next → Next Topic 17.21: CORS Middleware Configuration

17.21 CORS Middleware Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from fastapi.middleware.cors import CORSMiddleware` • Explicit `allow_origins` list • `allow_methods` • `allow_headers` • Production Edge-Case & Pitfall Guard (CORS Middleware Configuration) • Hands-On Python Implementation & Verification (CORS Middleware Configuration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CORS Middleware Configuration

Purpose: Allowing approved frontend origins to read custom `X-Request-ID` / pagination headers safely.

What Topic Is Next → Next Topic 17.22: Automatic OpenAPI 3 Schema Generation

17.22 Automatic OpenAPI 3 Schema Generation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Customizing `FastAPI(title=... • Version="1.0.0" • Openapi_tags=...)` • `responses={404: {"model": ErrorOut} • Production Edge-Case & Pitfall Guard (Automatic OpenAPI 3 Schema Generation) • Hands-On Python Implementation & Verification (Automatic OpenAPI 3 Schema Generation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Automatic OpenAPI 3 Schema Generation

Purpose: Generating TypeScript frontend API clients (`openapi-typescript`) directly from `/openapi.json` with zero manual typing!

What Topic Is Next → Next Topic 17.23: Interactive API Testing with Swagger UI (/docs)

17.23 Interactive API Testing with Swagger UI (/docs) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing OAuth2/Bearer endpoints directly in `/docs` • `swagger_ui_parameters={"persistAuthorization": True}` • Request/response schema examples • Production Edge-Case & Pitfall Guard (Interactive API Testing with Swagger UI (/docs)) • Hands-On Python Implementation & Verification (Interactive API Testing with Swagger UI (/docs)) • Interactive API Testing with Swagger UI (/docs) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interactive API Testing with Swagger UI (/docs)

Purpose: Enabling `persistAuthorization: True` so developers don't have to re-paste JWT tokens on every browser refresh.

What Topic Is Next → Next Topic 17.24: API Reference Documentation with ReDoc (/redoc)

17.24 API Reference Documentation with ReDoc (/redoc) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Clean 3-pane `/redoc` reference docs • Markdown descriptions in route docstrings • Exporting static `openapi.json` artifacts in CI • Production Edge-Case & Pitfall Guard (API Reference Documentation with ReDoc (/redoc)) • Hands-On Python Implementation & Verification (API Reference Documentation with ReDoc (/redoc)) • API Reference Documentation with ReDoc (/redoc) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Reference Documentation with ReDoc (/redoc)

Purpose: Sharing clean, searchable `/redoc` partner integration documentation.

What Topic Is Next → Next Topic 17.25: Environment Configuration Fundamentals

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

Subtopics Covered: Centralizing `APP_ENV` • `CORS_ORIGINS` • `MAX_UPLOAD_MB` with startup validation • `@lru_cache` singleton injection via `Depends(get_settings)` • Production Edge-Case & Pitfall Guard (Environment Configuration Fundamentals) • Hands-On Python Implementation & Verification (Environment Configuration Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Environment Configuration Fundamentals

Purpose: Overriding `get_settings` effortlessly in `pytest` without mutating global `os.environ` state!

What Topic Is Next → Module 17 Capstone (Week 17 Lab Build: Type-Safe FastAPI & Pydantic v2 Catalog REST Service) → Next Topic 18.1: Modular FastAPI Application Architecture

Module 17 Hands-On Capstone Project (100 Marks): Week 17 Lab Build: Type-Safe FastAPI & Pydantic v2 Catalog REST Service — Build a fast ASGI Product & Category API using FastAPI, Uvicorn, strict Pydantic v2 `ConfigDict(extra="forbid")` input/output schemas, `Depends()` pagination/auth injection, chunked `UploadF...

MODULE 18 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 18: FastAPI Advanced Development and Database Integration

Build production-grade, database-backed asynchronous FastAPI backends: modular `APIRouter` architecture, Service + Repository layers, SQLAlchemy 2.0 integration, `yield` DB session dependencies, CRUD APIs, Alembic migrations, `async def` vs `def` concurrency rules, Async SQLAlchemy (`create_async_engine`, `AsyncSession`, `asyncpg`), `pydantic-settings`, OAuth2 Password Flow, JWT access/refresh auth, Argon2/Bcrypt hashing, RBAC dependency guards, pagination, dynamic filtering/sorting, `/api/v1` versioning, `httpx.AsyncClient`, global error handling, and structured JSON logging.

Objectives: ✓ Architect modular FastAPI services (`APIRouter` -> `Service` -> `Repository` -> `AsyncSession`) with zero SQL inside route handlers. • ✓ Integrate Async SQLAlchemy 2.0 (`asyncpg`, `AsyncSession`, `expire_on_commit=False`) and async-configured `Alembic` migrations cleanly. • ✓ Distinguish `async def` (runs on event loop — MUST ONLY call non-blocking `await` I/O!) vs plain `def` (FastAPI runs in threadpool!). • ✓ Secure FastAPI APIs with `pydantic-settings`, Argon2/Bcrypt (`pwdlib`), OAuth2/JWT tokens, and composable `Depends` RBAC guards. • ✓ Implement generic pagination, dynamic filtering/sorting, `/api/v1` router versioning, pooled `httpx.AsyncClient`, global error envelopes, and JSON logging.

18.1 Modular FastAPI Application Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Application factory `def create_app() -> FastAPI:` • Clean layering (`api` -> `services` -> `repositories` -> `models`) • Dependency inversion • Production Edge-Case & Pitfall Guard (Modular FastAPI Application Architecture) • Hands-On Python Implementation & Verification (Modular FastAPI Application Architecture) • Modular FastAPI Application Architecture — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Modular FastAPI Application Architecture

Purpose: Creating fresh isolated `create_app()` instances in `pytest` and keeping business logic 100% testable without HTTP.

What Topic Is Next → Next Topic 18.2: Feature Routing with APIRouter

18.2 Feature Routing with APIRouter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `APIRouter(prefix="/products" • Tags=["Products"])` • Aggregating into `api_v1_router` • Shared router-level dependencies/responses • Production Edge-Case & Pitfall Guard (Feature Routing with APIRouter) • Hands-On Python Implementation & Verification (Feature Routing with APIRouter) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Feature Routing with APIRouter

Purpose: Mounting `auth_router`, `users_router`, `products_router`, and `orders_router` cleanly under `/api/v1`.

What Topic Is Next → Next Topic 18.3: Service Layer Business Logic Separation

18.3 Service Layer Business Logic Separation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `OrderService` orchestrating inventory checks • Pricing math • Repository writes • Transaction commit/rollback cleanly • Production Edge-Case & Pitfall Guard (Service Layer Business Logic Separation) • Hands-On Python Implementation & Verification (Service Layer Business Logic Separation) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Layer Business Logic Separation

Purpose: Keeping route functions `< 15` lines (parse input -> `await service.place_order(...)` -> return output schema).

What Topic Is Next → Next Topic 18.4: Repository Pattern for Data Access

18.4 Repository Pattern for Data Access [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Encapsulating `AsyncSession` `select()` • `add()` • `flush()` • Eager `selectinload`/`joinedload` queries inside `ProductRepository` • Production Edge-Case & Pitfall Guard (Repository Pattern for Data Access) • Hands-On Python Implementation & Verification (Repository Pattern for Data Access) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Repository Pattern for Data Access

Purpose: Allowing unit tests to test `OrderService` pricing/stock edge cases in 2ms using an in-memory `FakeProductRepository`.

What Topic Is Next → Next Topic 18.5: SQLAlchemy 2.0 Integration with FastAPI

18.5 SQLAlchemy 2.0 Integration with FastAPI [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mapping SQLAlchemy `DeclarativeBase` models cleanly to Pydantic `ConfigDict(from_attributes=True)` schemas without lazy-load surprises • Production Edge-Case & Pitfall Guard (SQLAlchemy 2.0 Integration with FastAPI) • Hands-On Python Implementation & Verification (SQLAlchemy 2.0 Integration with FastAPI) • SQLAlchemy 2.0 Integration with FastAPI — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for SQLAlchemy 2.0 Integration with FastAPI • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Converting eager-loaded `Order` + `Order.items` entities seamlessly into `OrderOut` JSON responses.

What Topic Is Next → Next Topic 18.6: Request-Scoped Database Session Dependency (yield)

18.6 Request-Scoped Database Session Dependency (yield) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `async def get_db() -> AsyncIterator[AsyncSession]: async with AsyncSessionLocal() as session: yield session` deterministic cleanup • Production Edge-Case & Pitfall Guard (Request-Scoped Database Session Dependency (yield)) • Hands-On Python Implementation & Verification (Request-Scoped Database Session Dependency (yield)) • Request-Scoped Database Session Dependency (yield) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Request-Scoped Database Session Dependency (yield) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Guaranteeing every HTTP request gets its own isolated `AsyncSession` returned to the pool even on unhandled `500` errors.

What Topic Is Next → Next Topic 18.7: Database-Backed CRUD REST APIs

18.7 Database-Backed CRUD REST APIs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Implementing async `POST` (`201`) • `GET` list/detail (`200`/`404`) • `PATCH` (`exclude_unset=True`) • `DELETE` (`204`) catching `IntegrityError` -> `409 Conflict` • Production Edge-Case & Pitfall Guard (Database-Backed CRUD REST APIs) • Hands-On Python Implementation & Verification (Database-Backed CRUD REST APIs) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend

Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Database-Backed CRUD REST APIs

Purpose: Translating PostgreSQL unique SKU/email `IntegrityError` races cleanly into `409 Conflict` after `await session.rollback()`.

What Topic Is Next → Next Topic 18.8: Alembic Schema Migrations in FastAPI

18.8 Alembic Schema Migrations in FastAPI [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Async Alembic template (`alembic init -t async migrations`) • Importing all `models` into `migrations/env.py` so `--autogenerate` sees every table! • Production Edge-Case & Pitfall Guard (Alembic Schema Migrations in FastAPI) • Hands-On Python Implementation & Verification (Alembic Schema Migrations in FastAPI) • Alembic Schema Migrations in FastAPI — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Alembic Schema Migrations in FastAPI • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running `alembic upgrade head` automatically in CI and container entrypoint scripts.

What Topic Is Next → Next Topic 18.9: Async Route Functions (async def vs def)

18.9 Async Route Functions (async def vs def) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exact FastAPI concurrency execution rule: `async def` runs directly on the single event loop (non-blocking `await` ONLY!) • Plain `def` is dispatched to an external AnyIO worker threadpool • Production Edge-Case & Pitfall Guard (Async Route Functions (async def vs def)) • Hands-On Python Implementation & Verification (Async Route Functions (async def vs def)) • Async Route Functions (async def vs def) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Async Route Functions (async def vs def) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Using `async def` with `asyncpg`/`httpx`/`redis.asyncio`, and using plain `def` (or `asyncio.to_thread`) if calling a synchronous library!

What Topic Is Next → Next Topic 18.10: Async SQLAlchemy (AsyncSession and asyncpg)

18.10 Async SQLAlchemy (AsyncSession and asyncpg) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `create_async_engine("postgresql+asyncpg://..." • Pool_pre_ping=True)` • `async_sessionmaker(expire_on_commit=False)` • `await engine.dispose()` on shutdown • Production Edge-Case & Pitfall Guard (Async SQLAlchemy (AsyncSession and asyncpg)) • Hands-On Python Implementation & Verification (Async SQLAlchemy (AsyncSession and asyncpg)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Async SQLAlchemy (AsyncSession and asyncpg)

Purpose: Achieving high-concurrency non-blocking PostgreSQL queries via `postgresql+asyncpg://` and `expire_on_commit=False`.

What Topic Is Next → Next Topic 18.11: Typed Settings with pydantic-settings

18.11 Typed Settings with pydantic-settings [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `from pydantic_settings import BaseSettings • SettingsConfigDict` • `SecretStr` (prevents accidental secret printing!) • `.env` loading • Production Edge-Case & Pitfall Guard (Typed Settings with pydantic-settings) • Hands-On Python Implementation & Verification (Typed Settings with pydantic-settings) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Typed Settings with pydantic-settings

Purpose: Using `jwt_secret: SecretStr` so `print(settings)` prints `SecretStr('**********')` instead of leaking your signing key in logs!

What Topic Is Next → Next Topic 18.12: FastAPI Security Utilities

18.12 FastAPI Security Utilities [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `HTTPBearer` • `OAuth2PasswordBearer(tokenUrl="/api/v1/auth/token")` • `OAuth2PasswordRequestForm` • Automatic OpenAPI padlock integration • Production Edge-Case & Pitfall Guard (FastAPI Security Utilities) • Hands-On Python Implementation & Verification (FastAPI Security Utilities) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for FastAPI Security Utilities

Purpose: Wiring Swagger UI's `Authorize` modal seamlessly to `/api/v1/auth/token` via `OAuth2PasswordBearer`.

What Topic Is Next → Next Topic 18.13: OAuth2 Password Bearer Flow Fundamentals

18.13 OAuth2 Password Bearer Flow Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Form-encoded `username` + `password` -> `{"access_token": "..." • "token_type": "bearer"}` RFC 6749 contract vs OAuth2 Authorization Code + PKCE overview • Production Edge-Case & Pitfall Guard (OAuth2 Password Bearer Flow Fundamentals) • Hands-On Python Implementation & Verification (OAuth2 Password Bearer Flow Fundamentals) • OAuth2 Password Bearer Flow Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for OAuth2 Password Bearer Flow Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Supporting standard `OAuth2PasswordRequestForm` login plus understanding third-party Google/GitHub OAuth2 Authorization Code flow.

What Topic Is Next → Next Topic 18.14: Stateless JWT Access and Refresh Tokens

18.14 Stateless JWT Access and Refresh Tokens [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Encoding/verifying JWTs with `PyJWT` (`jwt.encode` / `jwt.decode(... • Algorithms=["HS256"])`) • `typ` claim (`"access"` vs `"refresh"`!) • `exp` • Production Edge-Case & Pitfall Guard (Stateless JWT Access and Refresh Tokens) • Hands-On Python Implementation & Verification (Stateless JWT Access and Refresh Tokens) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Stateless JWT Access

Purpose: Embedding `"typ": "access"` vs `"typ": "refresh"` and verifying `payload["typ"] == "access"` so a Refresh Token can NEVER be used directly as an API Bearer Access Token!

What Topic Is Next → Next Topic 18.15: Secure Password Hashing (Argon2 / Bcrypt)

18.15 Secure Password Hashing (Argon2 / Bcrypt) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Modern `pwdlib` (`PasswordHash.recommended()` using Argon2id • Recommended by FastAPI docs over unmaintained `passlib`!) • Offloading CPU-bound hashing via `asyncio.to_thread` • Production Edge-Case & Pitfall Guard (Secure Password Hashing (Argon2 / Bcrypt)) • Hands-On Python Implementation &

Verification (Secure Password Hashing (Argon2 / Bcrypt)) • Secure Password Hashing (Argon2 / Bcrypt) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secure Password Hashing (Argon2 / Bcrypt)

Purpose: Hashing & verifying passwords with `pwdlib` Argon2id inside `await asyncio.to_thread(password_hash.verify, plain, hashed)` so 100ms CPU hashing never blocks the async eve...

What Topic Is Next → Next Topic 18.16: Role-Based Authorization Checks

18.16 Role-Based Authorization Checks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Checking `user.is_active` + `user.role` (`customer` • `vendor` • `admin`) + resource ownership (`order.user_id == current_user.id`) cleanly • Production Edge-Case & Pitfall Guard (Role-Based Authorization Checks) • Hands-On Python Implementation & Verification (Role-Based Authorization Checks) • Role-Based Authorization Checks — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Role-Based Authorization Checks

Purpose: Preventing horizontal privilege escalation (`GET /api/v1/orders/{order_id}` returning `403`/`404` if `order.user_id != user.id`).

What Topic Is Next → Next Topic 18.17: Dependency-Based Security Guards

18.17 Dependency-Based Security Guards [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Composable `CurrentUser = Annotated[User • Depends(get_current_active_user)]` • `AdminUser = Annotated[User • Depends(require_roles(Role.ADMIN))]` type aliases • Production Edge-Case & Pitfall Guard (Dependency-Based Security Guards) • Hands-On Python Implementation & Verification (Dependency-Based Security Guards) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dependency-Based Security Guards

Purpose: Securing any route simply by declaring `current_admin: AdminUser` in its parameter signature!

What Topic Is Next → Next Topic 18.18: Paginated API Responses

18.18 Paginated API Responses [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generic `PaginatedResponse[T]` (`items: list[T]` • `total: int` • `limit: int` • `offset: int` • Production Edge-Case & Pitfall Guard (Paginated API Responses) • Hands-On Python Implementation & Verification (Paginated API Responses) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Paginated API Responses

Purpose: Returning typed `PaginatedResponse[ProductOut]` across every collection endpoint consistently.

What Topic Is Next → Next Topic 18.19: Dynamic Query Filtering

18.19 Dynamic Query Filtering [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Typed `ProductFilterParams` dependency building conditional `.where(...)` clauses (`category_id` • `in_stock` • `price_min` • `price_max` • Production Edge-Case & Pitfall Guard (Dynamic Query Filtering) • Hands-On Python Implementation & Verification (Dynamic Query Filtering) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Query Filtering

Purpose: Combining 5 optional search filters cleanly with zero raw SQL string concatenation.

What Topic Is Next → Next Topic 18.20: Dynamic Multi-Column Sorting

18.20 Dynamic Multi-Column Sorting [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mapping validated `Literal["created_at" • "price" • "name"]` sort keys + `Literal["asc" • "desc"]` to whitelisted SQLAlchemy column attributes (NEVER `getattr(Model • Production Edge-Case & Pitfall Guard (Dynamic Multi-Column Sorting) • Hands-On Python Implementation & Verification (Dynamic Multi-Column Sorting) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dynamic Multi-Column Sorting

Purpose: Allowing `?sort_by=price&sort_dir=asc` safely against an explicit column map `{ "price": Product.price, "created_at": Product.created_at }`.

What Topic Is Next → Next Topic 18.21: Versioned FastAPI Routers (/api/v1)

18.21 Versioned FastAPI Routers (/api/v1) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Prefixing `/api/v1` via `APIRouter` • Sub-application `app.mount("/api/v2" • App_v2)` option for separate OpenAPI docs • `Sunset`/`Deprecation` headers • Production Edge-Case & Pitfall Guard (Versioned FastAPI Routers (/api/v1)) • Hands-On Python Implementation & Verification (Versioned FastAPI Routers (/api/v1)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Versioned FastAPI Routers (/api/v1)

Purpose: Keeping `/healthz` unversioned at root for load balancers while isolating business routes under `/api/v1/...`.

What Topic Is Next → Next Topic 18.22: External API Integration with httpx.AsyncClient

18.22 External API Integration with httpx.AsyncClient [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Creating ONE pooled `httpx.AsyncClient(timeout=httpx.Timeout(5.0 • Connect=2.0))` in `app.state` via `lifespan` • Injecting it via `Depends(get_http_client)` • Production Edge-Case & Pitfall Guard (External API Integration with httpx.AsyncClient) • Hands-On Python Implementation & Verification (External API Integration with httpx.AsyncClient) • External API Integration with httpx.AsyncClient — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for External API Integration with httpx.AsyncClient

Purpose: Reusing persistent HTTP/1.1 & HTTP/2 keep-alive connections across thousands of inter-service calls.

What Topic Is Next → Next Topic 18.23: Global Exception Handling Middleware

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

Subtopics Covered: Unified RFC 7807 Problem Details / structured error JSON across `DomainError` • `HTTPException` • `RequestValidationError` • Catch-all `500` handler (preserving `X-Request-ID`) • Production Edge-Case & Pitfall Guard (Global Exception Handling Middleware) • Hands-On Python Implementation & Verification (Global Exception Handling Middleware) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Global Exception Handling Middleware

Purpose: Guaranteeing unhandled `500` errors log full tracebacks internally with `request_id` while returning safe `{"code": "internal_error", "request_id": "..."}` externally.

What Topic Is Next → Next Topic 18.24: Structured JSON Application Logging

18.24 Structured JSON Application Logging [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Formatting every log record as single-line JSON (`timestamp` • `level` • `service` • `request_id` • Production Edge-Case & Pitfall Guard (Structured JSON Application Logging) • Hands-On Python Implementation & Verification (Structured JSON Application Logging) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Structured JSON Application Logging

Purpose: Querying all logs across 6 microservices for a single checkout via `request_id="req-8821"` in Kibana/CloudWatch.

What Topic Is Next → Module 18 Capstone (Week 18 Lab Build: Async SQLAlchemy 2.0 + JWT RBAC Order Backend) → Next Topic 19.1: Monolithic Application Design

Module 18 Hands-On Capstone Project (100 Marks): Week 18 Lab Build: Async SQLAlchemy 2.0 + JWT RBAC Order Backend — Connect FastAPI non-blockingly to PostgreSQL via `asyncpg` + `AsyncSession(expire_on_commit=False)`, async `Alembic`, `pydantic-settings` (`SecretStr`), `pwdlib` Argon2 + `PyJWT` access/refr...

MODULE 19 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 19: Microservices Architecture and Distributed Systems

Understand when and how to decompose a modular monolith into resilient microservices without hype: monolith vs microservices trade-offs, bounded contexts & DDD basics, independent deployment, sync REST vs `gRPC` fundamentals, API Gateway, service discovery, distributed config, Database-per-Service ownership, distributed transaction challenges, Saga pattern, event-driven architecture, message brokers, retries with jitter, timeouts, circuit breakers, idempotency keys, observability (logs/metrics/traces), correlation IDs, and service-to-service auth.

Objectives: ✓ Evaluate Modular Monolith vs Microservices trade-offs honestly and identify clean Domain-Driven Bounded Contexts. • ✓ Choose accurately between synchronous HTTP/REST, `gRPC` RPCs, and asynchronous Event-Driven Broker messaging. • ✓ Enforce Database-per-Service data ownership and coordinate multi-service workflows using Choreography/Orchestration Sagas with compensating actions. • ✓ Prevent cascading distributed outages using strict Timeouts, Exponential Backoff + Jitter Retries, Circuit Breakers, and `Idempotency-Key` deduplication. • ✓ Trace requests end-to-end across services with `X-Correlation-ID`, Prometheus/OpenTelemetry observability fundamentals, and internal JWT/mTLS auth.

19.1 Monolithic Application Design [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Well-structured Modular Monolith strengths: single ACID database transaction • In-memory function calls (`ns` vs `ms` network) • Simple deployment & debugging • Production Edge-Case & Pitfall Guard (Monolithic Application Design) • Hands-On Python Implementation & Verification (Monolithic Application Design) • Monolithic Application Design — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Monolithic Application Design

Purpose: Recognizing why Shopify, GitHub, and Stack Overflow scaled massive traffic on modular monoliths before extracting targeted services.

What Topic Is Next → Next Topic 19.2: Microservices and Service-Oriented Architecture

19.2 Microservices and Service-Oriented Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Independently deployable • Business-capability-aligned services owning their own data • Communicating over well-defined network contracts • Production Edge-Case & Pitfall Guard (Microservices and Service-Oriented Architecture) • Hands-On Python Implementation & Verification (Microservices and Service-Oriented Architecture) • Microservices and Service-Oriented Architecture — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Microservices

Purpose: Scaling `SearchService` or `NotificationService` to 10 replicas independently during hiring/sale traffic spikes.

What Topic Is Next → Next Topic 19.3: Monolith vs Microservices Architectural Trade-Offs

19.3 Monolith vs Microservices Architectural Trade-Offs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Fallacies of Distributed Computing: network latency • Partial failures • Eventual consistency • Distributed debugging • Production Edge-Case & Pitfall Guard (Monolith vs Microservices Architectural Trade-Offs) • Hands-On Python Implementation & Verification (Monolith vs Microservices Architectural Trade-Offs) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Monolith vs Microservices Architectural Trade-Offs

Purpose: Answering senior backend interview trade-off questions with balanced engineering judgment rather than buzzwords.

What Topic Is Next → Next Topic 19.4: Service Boundaries and Bounded Contexts

19.4 Service Boundaries and Bounded Contexts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Cutting boundaries along business capabilities (high internal cohesion • Low external chatty coupling) rather than technical layers • Production Edge-Case & Pitfall Guard (Service Boundaries and Bounded Contexts) • Hands-On Python Implementation & Verification (Service Boundaries and Bounded Contexts) • Service Boundaries and Bounded Contexts — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Boundaries • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Separating `Catalog Product` (title, images, specs) from `Warehouse Inventory` (stock_on_hand, reserved_qty) cleanly.

What Topic Is Next → Next Topic 19.5: Domain-Driven Design (DDD) Fundamentals

19.5 Domain-Driven Design (DDD) Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ubiquitous Language • Entities (identity) • Value Objects (`Money` • `Address`) • Production Edge-Case & Pitfall Guard (Domain-Driven Design (DDD) Fundamentals) • Hands-On Python Implementation & Verification (Domain-Driven Design (DDD) Fundamentals) • Idiomatic PBCP Implementation: clean modular

architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Domain-Driven Design (DDD) Fundamentals

Purpose: Meaning of `"User"` differs by Bounded Context: `AuthContext` cares about credentials/roles; `ShippingContext` cares about recipient name & address.

What Topic Is Next → Next Topic 19.6: Independent Service Deployment

19.6 Independent Service Deployment [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Backward-compatible API & event schema evolution (Expand- • -Contract) • Independent Git/CI pipelines • Zero lockstep coordinated releases • Production Edge-Case & Pitfall Guard (Independent Service Deployment) • Hands-On Python Implementation & Verification (Independent Service Deployment) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Independent Service Deployment

Purpose: Deploying `product-service` v1.4 without redeploying or breaking `order-service` v1.2.

What Topic Is Next → Next Topic 19.7: Synchronous Inter-Service Communication

19.7 Synchronous Inter-Service Communication [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Request-response blocking chains (`Gateway -> Order -> Inventory -> Payment`) • Latency addition • Availability multiplication (`0.99^4 = 96%`!) • Production Edge-Case & Pitfall Guard (Synchronous Inter-Service Communication) • Hands-On Python Implementation & Verification (Synchronous Inter-Service Communication) • Synchronous Inter-Service Communication — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Synchronous Inter-Service Communication

Purpose: Using synchronous calls ONLY when immediate real-time query/validation response is strictly required.

What Topic Is Next → Next Topic 19.8: HTTP/REST Service Integration

19.8 HTTP/REST Service Integration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Internal REST contracts with `httpx.AsyncClient` • Typed Pydantic client response validation • Connection pooling • Explicit connect/read timeouts • Production Edge-Case & Pitfall Guard (HTTP/REST Service Integration) • Hands-On Python Implementation & Verification (HTTP/REST Service Integration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for HTTP/REST Service Integration

Purpose: Validating `InventoryReserveResponse.model_validate(resp.json())` when `order-service` calls `inventory-service`.

What Topic Is Next → Next Topic 19.9: gRPC and Protocol Buffers Fundamentals

19.9 gRPC and Protocol Buffers Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Contract-first `.proto` schemas • Binary Protobuf serialization • HTTP/2 multiplexing • Generated Python stubs • Production Edge-Case & Pitfall Guard (gRPC and Protocol Buffers Fundamentals) • Hands-On Python Implementation & Verification (gRPC and Protocol Buffers Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for gRPC

Purpose: Choosing gRPC for ultra-low-latency internal east-west calls while keeping REST/JSON on public north-south browser/mobile APIs.

What Topic Is Next → Next Topic 19.10: API Gateway Architecture

19.10 API Gateway Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single north-south entry point (Nginx / Kong / Traefik / FastAPI Gateway): path routing • TLS termination • JWT edge verification • Rate limiting • Production Edge-Case & Pitfall Guard (API Gateway Architecture) • Hands-On Python Implementation & Verification (API Gateway Architecture) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for API Gateway Architecture

Purpose: Routing `/api/v1/users/*` -> `user-service:8001` and `/api/v1/orders/*` -> `order-service:8004` behind one public HTTPS domain.

What Topic Is Next → Next Topic 19.11: Service Registration and Discovery Fundamentals

19.11 Service Registration and Discovery Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dynamic container IP resolution via Docker Compose DNS / Kubernetes CoreDNS (`http://inventory-service:8003`) vs client-side registry (Consul/Eureka) concepts • Production Edge-Case & Pitfall Guard (Service Registration and Discovery Fundamentals) • Hands-On Python Implementation & Verification (Service Registration and Discovery Fundamentals) • Service Registration and Discovery Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service Registration • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Resolving `http://inventory-service:8000` via container DNS + environment URLs without hardcoding `172.18.0.5` container IPs!

What Topic Is Next → Next Topic 19.12: Centralized and Distributed Configuration

19.12 Centralized and Distributed Configuration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: 12-Factor environment injection • ConfigMaps/Secrets • Startup schema validation via `pydantic-settings` • Dynamic feature-flag concepts • Production Edge-Case & Pitfall Guard (Centralized and Distributed Configuration) • Hands-On Python Implementation & Verification (Centralized and Distributed Configuration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Centralized

Purpose: Validating every service's required broker/DB/downstream URLs at container boot before accepting traffic on `/readyz`.

What Topic Is Next → Next Topic 19.13: Data Ownership (Database-per-Service)

19.13 Data Ownership (Database-per-Service) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Strict rule: ONLY `inventory-service` reads/writes `inventory_db`! Other services access inventory strictly via API or published domain events • Production Edge-Case & Pitfall Guard (Data Ownership (Database-per-Service)) • Hands-On Python Implementation & Verification (Data Ownership (Database-per-Service)) • Data Ownership (Database-per-Service) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Data Ownership (Database-per-Service) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling &

Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Allowing `inventory-service` to migrate its schema or add indexes anytime without breaking `order-service` SQL queries!

What Topic Is Next → Next Topic 19.14: Distributed Transaction Consistency Challenges

19.14 Distributed Transaction Consistency Challenges [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why 2-Phase Commit (`2PC` / XA) locks struggle across independent services/brokers • Eventual consistency • The Dual-Write Problem (DB commit succeeds • Kafka publish fails!) • Production Edge-Case & Pitfall Guard (Distributed Transaction Consistency Challenges) • Hands-On Python Implementation & Verification (Distributed Transaction Consistency Challenges) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Distributed Transaction Consistency Challenges

Purpose: Solving the Dual-Write Problem via the Transactional Outbox Pattern (writing `Order` + `OutboxEvent` in the SAME PostgreSQL ACID transaction!).

What Topic Is Next → Next Topic 19.15: Saga Pattern (Choreography and Orchestration)

19.15 Saga Pattern (Choreography and Orchestration) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sequencing local ACID transactions across services (`Order PENDING -> Reserve Stock -> Charge Payment -> Order CONFIRMED`) + compensating undo actions (`Release Stock` • `Cancel Order`) on failure! • Production Edge-Case & Pitfall Guard (Saga Pattern (Choreography and Orchestration)) • Hands-On Python Implementation & Verification (Saga Pattern (Choreography and Orchestration)) • Saga Pattern (Choreography and Orchestration) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Saga Pattern (Choreography and Orchestration) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Automatically releasing reserved inventory and marking `Order = PAYMENT_FAILED` when `PaymentService` declines a card!

What Topic Is Next → Next Topic 19.16: Event-Driven Architecture

19.16 Event-Driven Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Decoupled producers & consumers (`OrderPlaced` • `StockReserved` • `PaymentCaptured`) • Event Notification vs Event-Carried State Transfer • Production Edge-Case & Pitfall Guard (Event-Driven Architecture) • Hands-On Python Implementation & Verification (Event-Driven Architecture) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event-Driven Architecture

Purpose: Letting `NotificationService`, `AnalyticsService`, and `LoyaltyService` react to `OrderConfirmed` independently without `OrderService` even knowing they exist!

What Topic Is Next → Next Topic 19.17: Message Broker Fundamentals

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

Subtopics Covered: Smart broker / flexible routing (`RabbitMQ` AMQP task/work queues) vs distributed append-only commit log (`Apache Kafka` high-throughput replayable event streams) • Production Edge-Case & Pitfall Guard (Message Broker Fundamentals) • Hands-On Python Implementation & Verification (Message Broker Fundamentals) • Message Broker Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Message Broker Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Picking RabbitMQ + Celery for task dispatch/retries and Apache Kafka when multiple consumer groups need ordered, replayable event streams.

What Topic Is Next → Next Topic 19.18: Retry Mechanisms with Exponential Backoff

19.18 Retry Mechanisms with Exponential Backoff [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Retrying ONLY transient errors (`502/503/504` • `ConnectTimeout`) — NEVER `400/401/403/404/422`! — with exponential backoff `2^attempt` + random jitter (`tenacity`) • Production Edge-Case & Pitfall Guard (Retry Mechanisms with Exponential Backoff) • Hands-On Python Implementation & Verification (Retry Mechanisms with Exponential Backoff) • Retry Mechanisms with Exponential Backoff — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Retry Mechanisms with Exponential Backoff • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Adding random jitter so 1,000 failed clients don’t hammer a recovering service at the exact same millisecond ("thundering herd")!

What Topic Is Next → Next Topic 19.19: Request Timeouts and Deadline Propagation

19.19 Request Timeouts and Deadline Propagation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit connect timeout (`2s`) + read timeout (`3s`) on EVERY outbound socket (HTTP • Redis • SMTP) so hung dependencies never exhaust worker pools • Production Edge-Case & Pitfall Guard (Request Timeouts and Deadline Propagation) • Hands-On Python Implementation & Verification (Request Timeouts and Deadline Propagation) • Request Timeouts and Deadline Propagation — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Request Timeouts

Purpose: Failing fast in `3.0s` when `payment-gateway` hangs instead of tying up all Uvicorn coroutines/DB connections.

What Topic Is Next → Next Topic 19.20: Circuit Breaker Failure Isolation

19.20 Circuit Breaker Failure Isolation [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: State machine: `CLOSED` (normal) -> failure threshold exceeded -> `OPEN` (fail fast `< 1ms` without calling dead downstream!) -> cooldown -> `HALF_OPEN` probe -> `CLOSED` • Production Edge-Case & Pitfall Guard (Circuit Breaker Failure Isolation) • Hands-On Python Implementation & Verification (Circuit Breaker Failure Isolation) • Circuit Breaker Failure Isolation — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Circuit Breaker Failure Isolation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Opening the circuit when `recommendation-service` has 5 consecutive timeouts, returning fallback popular items in `1ms` while giving the downstream service breathing room...

What Topic Is Next → Next Topic 19.21: Idempotency Keys for Safe Retries

19.21 Idempotency Keys for Safe Retries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Storing `Idempotency-Key` (`UNIQUE (user_id • Idempotency_key)`) + request payload hash + cached response in PostgreSQL/Redis • `processed_events(event_id PK)` for broker consumers • Production Edge-Case & Pitfall Guard (Idempotency Keys for Safe Retries) • Hands-On Python Implementation & Verification (Idempotency Keys for Safe Retries) • Idempotency Keys for Safe Retries — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Idempotency Keys for Safe Retries

Purpose: Guaranteeing that clicking "Pay Now" twice or retrying a timed-out `POST /orders` returns the exact same `Order #1042` without double-charging!

What Topic Is Next → Next Topic 19.22: Observability Pillars (Logs, Metrics, Traces)

19.22 Observability Pillars (Logs, Metrics, Traces) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Three pillars: Structured Logs (what happened) • Metrics — RED rate/errors/duration histograms (`prometheus-client`) • Distributed Traces (`OpenTelemetry` spans) • Production Edge-Case & Pitfall Guard (Observability Pillars (Logs, Metrics, Traces)) • Hands-On Python Implementation & Verification (Observability Pillars (Logs, Metrics, Traces)) • Observability Pillars (Logs, Metrics, Traces) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Observability Pillars (Logs, Metrics, Traces)

Purpose: Exposing `/metrics` request latency histograms (`p50`, `p95`, `p99`) and error rates per route.

What Topic Is Next → Next Topic 19.23: Correlation IDs for Distributed Request Tracing

19.23 Correlation IDs for Distributed Request Tracing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Generating `X-Correlation-ID` at the API Gateway • Forwarding it automatically on every outbound `httpx` header • Every Celery/RabbitMQ/Kafka message header! • Production Edge-Case & Pitfall Guard (Correlation IDs for Distributed Request Tracing) • Hands-On Python Implementation & Verification (Correlation IDs for Distributed Request Tracing) • Correlation IDs for Distributed Request Tracing — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Correlation IDs for Distributed Request Tracing

Purpose: Reconstructing the complete timeline of `Gateway -> OrderService -> Kafka -> InventoryService -> NotificationService` with one correlation ID.

What Topic Is Next → Next Topic 19.24: Service-to-Service Authentication

19.24 Service-to-Service Authentication [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zero-trust internal security: verifying `aud` (audience) + `iss` (issuer) claims on internal service JWTs • HMAC webhook signatures (`hmac.compare_digest`) • Private Docker/VPC network isolation • Production Edge-Case & Pitfall Guard (Service-to-Service Authentication) • Hands-On Python Implementation & Verification (Service-to-Service Authentication) • Service-to-Service Authentication — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Service-to-Service Authentication

Purpose: Ensuring only `order-service` (not arbitrary pods or external callers) can invoke internal `POST /internal/inventory/reserve`.

What Topic Is Next → Module 19 Capstone (Week 19 Lab Build: Resilient Two-Service Saga, Circuit Breaker & Idempotency Mesh) → Next Topic 20.1: Redis In-Memory Data Store Introduction

Module 19 Hands-On Capstone Project (100 Marks): Week 19 Lab Build: Resilient Two-Service Saga, Circuit Breaker & Idempotency Mesh — Connect `order-service` and `inventory-service` behind a lightweight API Gateway enforcing Database-per-Service, `Idempotency-Key` deduplication, Transactional Outbox, Circuit Breaker state...

MODULE 20 OF 24 • STAGE 5 • 25 LIVE + LAB HOURS • 28 TOPICS (224 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 20: Redis, Celery, RabbitMQ and Kafka

Master in-memory caching, distributed background workers, message brokers, and event streaming in Python — culminating in Major Project 5: Redis data types & commands, application caching, TTLs, cache invalidation, Cache-Aside, Redis distributed locks, Celery workers, tasks, Celery Beat scheduling, retries & result backends, RabbitMQ exchanges (`Direct`, `Topic`, `Fanout`), queues, routing keys, acks & Dead-Letter Queues (DLQs), Apache Kafka brokers/topics/partitions, producers, consumers, consumer groups, offsets, event-driven service integration, and Major Project 5 (`E-Commerce Microservices Backend`).

Objectives: ✓ Use Redis 7 (`redis.asyncio` / `redis-py`) fluently for sub-millisecond Cache-Aside caching, TTL jitter, invalidation, rate limiting, and distributed locks. • ✓ Build durable background & scheduled jobs with Celery Workers, Celery Beat, exponential retries, and result backends. • ✓ Route asynchronous workflows reliably over RabbitMQ AMQP (`Direct`, `Topic`, `Fanout` exchanges, manual `ack`, and Dead-Letter Queues). • ✓ Publish and consume partitioned event streams with Apache Kafka (partition keys for per-order ordering, consumer groups, and manual offset commits). • ✓ Deliver Major Project 5: a 6-service FastAPI E-Commerce Microservices Backend integrating PostgreSQL, SQLAlchemy, Redis, JWT, and RabbitMQ/Kafka.

20.1 Redis In-Memory Data Store Introduction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Single-threaded event-loop command execution (atomic individual commands!) • Sub-millisecond RAM latency • RDB snapshots vs AOF persistence • Eviction policies (`allkeys-lru`) • Production Edge-Case & Pitfall Guard (Redis In-Memory Data Store Introduction) • Hands-On Python Implementation & Verification (Redis In-Memory Data Store Introduction) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Redis In-Memory Data Store Introduction

Purpose: Serving hot product catalogs in `0.4ms` from Redis instead of hitting PostgreSQL 5,000 times per second.

What Topic Is Next → Next Topic 20.2: Redis Data Types (Strings, Lists, Sets, Hashes, Sorted Sets)

20.2 Redis Data Types (Strings, Lists, Sets, Hashes, Sorted Sets) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `STRING` (JSON cache • Atomic `INCR`) • `HASH` (field maps) • `LIST` (queues) • Production Edge-Case & Pitfall Guard (Redis Data Types (Strings, Lists, Sets, Hashes, Sorted Sets)) • Hands-On Python Implementation & Verification (Redis Data Types (Strings, Lists, Sets, Hashes, Sorted Sets)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production

Runtime Execution & Memory Lifecycle for Redis Data Types (Strings, Lists, Sets, Hashes, Sort...

Purpose: Using `ZSET` (`ZADD`, `ZREMRANGEBYSCORE`, `ZCARD`) for exact sliding-window API rate limiting and trending job leaderboards.

What Topic Is Next → Next Topic 20.3: Core Redis CLI and redis-py Commands

20.3 Core Redis CLI and redis-py Commands [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `redis.asyncio.from_url(REDIS_URL • Decode_responses=True)` • Connection pooling • `pipeline()` batching multiple commands in 1 network roundtrip • Production Edge-Case & Pitfall Guard (Core Redis CLI and redis-py Commands) • Hands-On Python Implementation & Verification (Core Redis CLI and redis-py Commands) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Core Redis CLI

Purpose: Executing `INCR` + `EXPIRE` in a single atomic roundtrip via `async with r.pipeline(transaction=True) as pipe:`.

What Topic Is Next → Next Topic 20.4: Application Response and Query Caching

20.4 Application Response and Query Caching [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Namespaced versioned cache keys (`prod:v1:product:{id}`) • JSON Pydantic `model_dump_json()` / `model_validate_json()` • Measuring cache hit ratio • Production Edge-Case & Pitfall Guard (Application Response and Query Caching) • Hands-On Python Implementation & Verification (Application Response and Query Caching) • Application Response and Query Caching — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Application Response

Purpose: Caching `prod:v1:product:42` serialized JSON so `ProductService.get_product(42)` skips SQLAlchemy on cache hits.

What Topic Is Next → Next Topic 20.5: Cache Expiration and TTL Management

20.5 Cache Expiration and TTL Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mandatory `ex=` TTL on every cache key • TTL jitter (`300 + random.randint(0 • 45)`) preventing Cache Avalanche (mass simultaneous expiration!) • Caching negative `"NOT_FOUND"` sentinels briefly (`30s`) • Production Edge-Case & Pitfall Guard (Cache Expiration and TTL Management) • Hands-On Python Implementation & Verification (Cache Expiration and TTL Management) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cache Expiration

Purpose: Adding random `0–45s` TTL jitter across 10,000 warmed product keys so they don't all expire at the exact same second and crush PostgreSQL!

What Topic Is Next → Next Topic 20.6: Cache Invalidation and Consistency

20.6 Cache Invalidation and Consistency [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Explicit `await redis.delete(key)` AFTER PostgreSQL commit on `PATCH`/`DELETE` (delete rather than overwrite to avoid concurrent write-order races!) • Production Edge-Case & Pitfall Guard (Cache Invalidation and Consistency) • Hands-On Python Implementation & Verification (Cache Invalidation and Consistency) • Cache Invalidation and Consistency — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cache Invalidation • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Updating product price in PostgreSQL, committing, and deleting `ecom:v1:product:{id}` so the next read populates fresh truth.

What Topic Is Next → Next Topic 20.7: Cache-Aside Strategy

20.7 Cache-Aside Strategy [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Read-through vs Write-through vs lazy Cache-Aside (`1. Check Redis -> 2. Miss: Query DB -> 3. SET Redis with TTL`) + single-flight lock against Cache Stampede on hot keys • Production Edge-Case & Pitfall Guard (Cache-Aside Strategy) • Hands-On Python Implementation & Verification (Cache-Aside Strategy) • Cache-Aside Strategy — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Cache-Aside Strategy • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Protecting a viral flash-sale product key from 500 concurrent cache-miss DB queries using a short stampede lock.

What Topic Is Next → Next Topic 20.8: Distributed Locks with Redis

20.8 Distributed Locks with Redis [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `SET lock_key unique_token NX PX 10000` acquisition • Atomic Lua compare- • -delete release (`if redis.call("get" • KEYS[1]) == ARGV[1] then return redis.call("del" • Production Edge-Case & Pitfall Guard (Distributed Locks with Redis) • Hands-On Python Implementation & Verification (Distributed Locks with Redis) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Distributed Locks with Redis

Purpose: Ensuring only ONE replica among 4 containers executes a scheduled settlement reconciliation job at a time.

What Topic Is Next → Next Topic 20.9: Celery Distributed Task Queue Architecture

20.9 Celery Distributed Task Queue Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Producer (FastAPI/Django) -> Message Broker (Redis/RabbitMQ) -> Celery Worker Pool -> Result Backend architecture • Production Edge-Case & Pitfall Guard (Celery Distributed Task Queue Architecture) • Hands-On Python Implementation & Verification (Celery Distributed Task Queue Architecture) • Celery Distributed Task Queue Architecture — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Celery Distributed Task Queue Architecture • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Offloading invoice PDF generation, bulk CSV imports, and email notifications outside the HTTP request path.

What Topic Is Next → Next Topic 20.10: Celery Worker Configuration and Concurrency

20.10 Celery Worker Configuration and Concurrency [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Prefork (`--concurrency=4`) vs I/O pools • `task_acks_late = True` + `worker_prefetch_multiplier = 1` for fair • Crash-safe long-task distribution • Production Edge-Case & Pitfall Guard (Celery Worker Configuration and Concurrency) • Hands-On Python Implementation & Verification (Celery Worker Configuration

and Concurrency) • Celery Worker Configuration and Concurrency — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Celery Worker Configuration

Purpose: Enabling `task_acks_late = True` so if a container is OOM-killed mid-task, the broker redelivers the unacknowledged task to a healthy worker!

What Topic Is Next → Next Topic 20.11: Defining Background Celery Tasks

20.11 Defining Background Celery Tasks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Idempotent task functions taking primitive JSON arguments (`order_id: str`) • Explicit `soft_time_limit` / `time_limit` • Structured task correlation logging • Production Edge-Case & Pitfall Guard (Defining Background Celery Tasks) • Hands-On Python Implementation & Verification (Defining Background Celery Tasks) • Defining Background Celery Tasks — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Defining Background Celery Tasks

Purpose: Generating and emailing an Order Invoice PDF asynchronously when `order_confirmed` fires.

What Topic Is Next → Next Topic 20.12: Periodic Task Scheduling with Celery Beat

20.12 Periodic Task Scheduling with Celery Beat [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `beat_schedule` (`crontab(minute="*/15")`) • Expiring unpaid pending orders after 30 minutes • Running STRICTLY ONE `celery beat` scheduler instance! • Production Edge-Case & Pitfall Guard (Periodic Task Scheduling with Celery Beat) • Hands-On Python Implementation & Verification (Periodic Task Scheduling with Celery Beat) • Periodic Task Scheduling with Celery Beat — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Periodic Task Scheduling with Celery Beat

Purpose: Releasing stale unpaid cart/inventory reservations every 5 minutes via a scheduled Celery Beat task.

What Topic Is Next → Next Topic 20.13: Task Retries and Failure Backoff

20.13 Task Retries and Failure Backoff [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `autoretry_for=(httpx.TransportError • `retry_backoff=True` • `retry_jitter=True` • `max_retries=5` • Production Edge-Case & Pitfall Guard (Task Retries and Failure Backoff) • Hands-On Python Implementation & Verification (Task Retries and Failure Backoff) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Task Retries

Purpose: Retrying webhook/SMTP delivery automatically across `1s -> 2s -> 4s -> 8s -> 16s` with jitter before marking `FAILED` in DB.

What Topic Is Next → Next Topic 20.14: Celery Result Backends

20.14 Celery Result Backends [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AsyncResult(task_id)` polling (`PENDING` • `STARTED` • `SUCCESS` • `FAILURE`) • Production Edge-Case & Pitfall Guard (Celery Result Backends) • Hands-On Python Implementation & Verification (Celery Result Backends) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Celery Result Backends

Purpose: Returning `202 Accepted {"job_id": task.id}` on heavy CSV export and letting clients poll `GET /exports/{job_id}`.

What Topic Is Next → Next Topic 20.15: RabbitMQ Message Broker Architecture

20.15 RabbitMQ Message Broker Architecture [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: AMQP 0-9-1 model: Producer -> Exchange -> Binding (Routing Key) -> Durable Queue -> Consumer • Virtual hosts (`vhost`) • Management UI (`:15672`) • Production Edge-Case & Pitfall Guard (RabbitMQ Message Broker Architecture) • Hands-On Python Implementation & Verification (RabbitMQ Message Broker Architecture) • RabbitMQ Message Broker Architecture — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for RabbitMQ Message Broker Architecture

Purpose: Understanding that RabbitMQ Producers NEVER publish directly to a queue — they publish to an Exchange which routes to bound Queues!

What Topic Is Next → Next Topic 20.16: AMQP Exchanges (Direct, Topic, Fanout)

20.16 AMQP Exchanges (Direct, Topic, Fanout) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Direct` (exact routing key) • `Topic` (wildcard `order.*` • `#.failed` pattern matching!) • `Fanout` (broadcast to ALL bound queues ignoring routing key) • Production Edge-Case & Pitfall Guard (AMQP Exchanges (Direct, Topic, Fanout)) • Hands-On Python Implementation & Verification (AMQP Exchanges (Direct, Topic, Fanout)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for AMQP Exchanges (Direct, Topic, Fanout)

Purpose: Using a `Topic` exchange `domain.events` where `inventory_q` binds `order.created` and `audit_q` binds `order.#`!

What Topic Is Next → Next Topic 20.17: Durable Queues and Bindings

20.17 Durable Queues and Bindings [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `durable=True` queue declaration • `DeliveryMode.PERSISTENT` (`delivery_mode=2`) messages • Idempotent startup topology declaration • `prefetch_count` • Production Edge-Case & Pitfall Guard (Durable Queues and Bindings) • Hands-On Python Implementation & Verification (Durable Queues and Bindings) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Durable Queues

Purpose: Surviving a RabbitMQ container restart with zero lost `order.created` messages (`durable=True` + `PERSISTENT` message!).

What Topic Is Next → Next Topic 20.18: Message Routing Keys

20.18 Message Routing Keys [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Designing hierarchical `<domain>.<entity>.<event>` routing keys (`ecom.order.placed` • `ecom.payment.captured`) for clean observability • Selective subscription • Production Edge-Case & Pitfall Guard (Message Routing Keys) • Hands-On Python Implementation & Verification (Message Routing Keys) • Message Routing Keys — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Message Routing Keys

Purpose: Allowing a new `analytics-service` to subscribe to `ecom.#` with zero code changes in `order-service` or `payment-service`!

What Topic Is Next → Next Topic 20.19: RabbitMQ Producers and Consumers (aio-pika / pika)

20.19 RabbitMQ Producers and Consumers (aio-pika / pika) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `aio_pika.connect_robust()` auto-reconnecting async channels • Publishing Pydantic JSON bytes with `message_id` & `correlation_id` • `queue.iterator()` consumption • Production Edge-Case & Pitfall Guard (RabbitMQ Producers and Consumers (aio-pika / pika)) • Hands-On Python Implementation & Verification (RabbitMQ Producers and Consumers (aio-pika / pika)) • RabbitMQ Producers and Consumers (aio-pika / pika) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for RabbitMQ Producers

Purpose: Publishing `OrderPlaced` events from async FastAPI via `aio-pika` robust connection pools.

What Topic Is Next → Next Topic 20.20: Message Acknowledgements (ack and nack)

20.20 Message Acknowledgements (ack and nack) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Manual acknowledgement AFTER DB commit (`async with message.process(requeue=False):`) vs `auto_ack=True` data-loss danger • `nack(requeue=True vs False)` • Production Edge-Case & Pitfall Guard (Message Acknowledgements (ack and nack)) • Hands-On Python Implementation & Verification (Message Acknowledgements (ack and nack)) • Message Acknowledgements (ack and nack) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Message Acknowledgements (ack and nack) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Sending `ack` ONLY after `NotificationService` commits the delivery record in PostgreSQL.

What Topic Is Next → Next Topic 20.21: Dead-Letter Queues (DLQ)

20.21 Dead-Letter Queues (DLQ) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `x-dead-letter-exchange` + `x-dead-letter-routing-key` on primary queues so rejected/expired messages land safely in `orders.dlq` for inspection & replay • Production Edge-Case & Pitfall Guard (Dead-Letter Queues (DLQ)) • Hands-On Python Implementation & Verification (Dead-Letter Queues (DLQ)) • Dead-Letter Queues (DLQ) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Dead-Letter Queues (DLQ) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Isolating failed poison messages in `orders.dlq` with alert monitoring while healthy orders continue processing uninterrupted.

What Topic Is Next → Next Topic 20.22: Apache Kafka Distributed Event Streaming Introduction

20.22 Apache Kafka Distributed Event Streaming Introduction [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Immutable append-only distributed commit log vs traditional queue (consuming a Kafka event does NOT delete it — retention is time/size-based • Enabling full replay!) • Production Edge-Case & Pitfall Guard (Apache Kafka Distributed Event Streaming Introduction) • Hands-On Python Implementation & Verification (Apache Kafka Distributed Event Streaming Introduction) • Apache Kafka Distributed Event Streaming Introduction — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Apache Kafka Distributed Event Streaming Introductio... • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Replaying the last 7 days of `order-events` from offset `0` to backfill a newly deployed `fraud-analytics` service!

What Topic Is Next → Next Topic 20.23: Kafka Architecture (Brokers, Topics, Partitions)

20.23 Kafka Architecture (Brokers, Topics, Partitions) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: KRaft mode (modern ZooKeeper-less Kafka!) • Brokers • Topics split into ordered Partitions (`P0 • P2`) • Production Edge-Case & Pitfall Guard (Kafka Architecture (Brokers, Topics, Partitions)) • Hands-On Python Implementation & Verification (Kafka Architecture (Brokers, Topics, Partitions)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Kafka Architecture (Brokers, Topics, Partitions)

Purpose: Using `key=order_id.encode()` so `OrderCreated`, `OrderPaid`, and `OrderShipped` for `ORD-99` ALWAYS land in the SAME partition in strict chronological order!

What Topic Is Next → Next Topic 20.24: Publishing Events with Kafka Producers

20.24 Publishing Events with Kafka Producers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `aiokafka.AIOKafkaProducer` / `confluent-kafka` • `acks="all"` durability • `enable_idempotence=True` • Batching (`linger_ms`) • Production Edge-Case & Pitfall Guard (Publishing Events with Kafka Producers) • Hands-On Python Implementation & Verification (Publishing Events with Kafka Producers) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Publishing Events with Kafka Producers

Purpose: Publishing durable `acks="all"` domain events from an Outbox relay or FastAPI service.

What Topic Is Next → Next Topic 20.25: Reading Events with Kafka Consumers

20.25 Reading Events with Kafka Consumers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `AIOKafkaConsumer` • `auto_offset_reset="earliest"` • Deserializing + validating Pydantic schemas • Error isolation (DLQ topic `<topic>.dlq`) • Production Edge-Case & Pitfall Guard (Reading Events with Kafka Consumers) • Hands-On Python Implementation & Verification (Reading Events with Kafka Consumers) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Reading Events with Kafka Consumers

Purpose: Consuming `order-events` in `inventory-service` and `notification-service` with `processed_events(event_id)` deduplication.

What Topic Is Next → Next Topic 20.26: Consumer Groups for Parallel Scaling

20.26 Consumer Groups for Parallel Scaling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `group_id` partition assignment: 1 partition is assigned to AT MOST 1 consumer within the same `group_id` (so `> num_partitions` consumers sit idle!) • Rebalance awareness • Production Edge-Case & Pitfall Guard (Consumer Groups for Parallel Scaling) • Hands-On Python Implementation & Verification

(Consumer Groups for Parallel Scaling) • Consumer Groups for Parallel Scaling — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Consumer Groups for Parallel Scaling • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Running `inventory-group` (3 consumers on 6 partitions) AND `notification-group` (2 consumers on the same 6 partitions) completely independently!

What Topic Is Next → Next Topic 20.27: Kafka Offsets and At-Least-Once Delivery

20.27 Kafka Offsets and At-Least-Once Delivery [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Disable `enable_auto_commit=False` • Call `await consumer.commit()` ONLY AFTER your PostgreSQL transaction commits (At-Least-Once + Idempotency)! • Production Edge-Case & Pitfall Guard (Kafka Offsets and At-Least-Once Delivery) • Hands-On Python Implementation & Verification (Kafka Offsets and At-Least-Once Delivery) • Kafka Offsets and At-Least-Once Delivery — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Kafka Offsets • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Ensuring zero lost events when a consumer pod restarts mid-processing.

What Topic Is Next → Next Topic 20.28: Event-Driven Integration Across FastAPI Services

20.28 Event-Driven Integration Across FastAPI Services [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Wiring the complete 6-service E-Commerce workflow (`User` • `Product` • `Inventory` • `Order` • Production Edge-Case & Pitfall Guard (Event-Driven Integration Across FastAPI Services) • Hands-On Python Implementation & Verification (Event-Driven Integration Across FastAPI Services) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Event-Driven Integration Across FastAPI Services

Purpose: Delivering Major Project 5 (`E-Commerce Microservices Backend`) with clean service boundaries, Redis caching, JWT auth, and async event-driven checkout!

What Topic Is Next → Module 20 Capstone (Project 5: E-Commerce Microservices Backend using FastAPI) → Next Topic 21.1: Software Testing Fundamentals

Module 20 Hands-On Capstone Project (100 Marks): Project 5: E-Commerce Microservices Backend using FastAPI — Architect and implement an independently structured 6-service E-Commerce Microservices Backend (`User Service`, `Product Service`, `Inventory Service`, `Order Service`, `Payment Simulation S...

MODULE 21 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 26 TOPICS (208 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 21: Python Testing and Quality Engineering

Make automated testing and static code quality non-negotiable engineering habits: testing pyramid, TDD, `unittest`, `pytest` setup, assertions & `pytest.raises`, `conftest.py` fixtures, `@pytest.mark.parametrize`, test doubles & `unittest.mock` (`Mock`, `MagicMock`, `AsyncMock`, `patch`), REST API testing across Django Test Client, DRF `APIClient`, and FastAPI `TestClient`/`httpx.AsyncClient`, isolated test DBs, integration/auth/RBAC/negative testing, `Coverage.py`, `Ruff`, `Black`, `mypy`, linting, and code review.

Objectives: ✓ Write fast, deterministic unit and integration suites using `unittest` and `pytest` (`conftest.py` fixtures, `@pytest.mark.parametrize`, `pytest.raises`). • ✓ Isolate external SMTP, payment gateway, and S3 dependencies cleanly using `unittest.mock.patch` and `AsyncMock` (patching where imported/looked up!). • ✓ Test Django/DRF (`APIClient`) and async FastAPI (`httpx.AsyncClient` + `ASGITransport` + transactional DB rollback) across `2xx`, `400`, `401`, `403`, `404`, `409`, and `422`. • ✓ Enforce `>= 80%` meaningful test coverage (`pytest-cov` / `Coverage.py`) including negative, auth, and cross-tenant RBAC boundary cases. • ✓ Automate formatting, linting, and static type gates with `Ruff`, `Black`, and `mypy` before every Git commit/PR.

21.1 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 Edge-Case & Pitfall Guard (Software Testing Fundamentals) • Hands-On Python Implementation & Verification (Software Testing Fundamentals) • Software Testing Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 21.2: Testing Pyramid (Unit, Integration, Functional)

21.2 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 Edge-Case & Pitfall Guard (Testing Pyramid (Unit, Integration, Functional)) • Hands-On Python Implementation & Verification (Testing Pyramid (Unit, Integration, Functional)) • Testing Pyramid (Unit, Integration, Functional) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Testing Pyramid (Unit, Integration, Functional) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

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 21.3: Test-Driven Development (TDD) Fundamentals

21.3 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 Edge-Case & Pitfall Guard (Test-Driven Development (TDD) Fundamentals) • Hands-On Python Implementation & Verification (Test-Driven Development (TDD) Fundamentals) • Test-Driven Development (TDD) Fundamentals — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Test-Driven Development (TDD) Fundamentals • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

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 21.4: Python Built-In unittest Framework

21.4 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 Edge-Case & Pitfall Guard (Python Built-In unittest Framework) • Hands-On Python Implementation & Verification (Python Built-In unittest Framework) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 21.5: Pytest Setup, Configuration and Execution

21.5 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 Edge-Case & Pitfall Guard (Pytest Setup, Configuration and Execution) • Hands-On Python Implementation & Verification (Pytest Setup, Configuration and Execution) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 21.6: Writing Expressive Pytest Test Functions

21.6 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 Edge-Case & Pitfall Guard (Writing Expressive Pytest Test Functions) • Hands-On Python Implementation & Verification (Writing Expressive Pytest Test Functions) • Writing Expressive Pytest Test Functions — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Writing Expressive Pytest Test Functions • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

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 21.7: Assertions and Exception Checks (pytest.raises)

21.7 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 Edge-Case & Pitfall Guard (Assertions and Exception Checks (pytest.raises)) • Hands-On Python Implementation & Verification (Assertions and Exception Checks (pytest.raises)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 21.8: Reusable Test State with Pytest Fixtures

21.8 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 Edge-Case & Pitfall Guard (Reusable Test State with Pytest Fixtures) • Hands-On Python Implementation & Verification (Reusable Test State with Pytest Fixtures) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime 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 21.9: Data-Driven Testing (@pytest.mark.parametrize)

21.9 Data-Driven Testing (@pytest.mark.parametrize) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running 12 boundary/invalid input cases cleanly via `@pytest.mark.parametrize("payload • Expected_status" • [...] • Ids=[...])` without copy-pasting 12 functions • Production Edge-Case & Pitfall Guard (Data-Driven Testing (@pytest.mark.parametrize)) • Hands-On Python Implementation & Verification (Data-Driven Testing (@pytest.mark.parametrize)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Data-Driven Testing (@pytest.mark.parametrize)

Purpose: Testing `0`, `-1`, `""`, `"not-an-email"`, `10_000_000`, SQLi strings, and Unicode names in one concise parametrized table.

What Topic Is Next → Next Topic 21.10: Test Doubles and Mocking Concepts

21.10 Test Doubles and Mocking Concepts [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Dummy • Fake (`FakePaymentGateway`) • Stub • Spy • Production Edge-Case & Pitfall Guard (Test Doubles and Mocking Concepts) • Hands-On Python Implementation & Verification (Test Doubles and Mocking Concepts) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Test Doubles

Purpose: Injecting a deterministic `FakePaymentGateway` or freezing `datetime` in unit tests so tests never charge real credit cards or hit external APIs.

What Topic Is Next → Next Topic 21.11: unittest.mock (Mock, MagicMock and patch)

21.11 unittest.mock (Mock, MagicMock and patch) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `patch("app.services.orders.payment_client" • Autospec=True)` • `MagicMock` • `AsyncMock` (for `await` coroutines!) • Production Edge-Case & Pitfall Guard (unittest.mock (Mock, MagicMock and patch)) • Hands-On Python Implementation & Verification (unittest.mock (Mock, MagicMock and patch)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for unittest.mock (Mock, MagicMock and patch)

Purpose: Simulating `side_effect=httpx.ConnectTimeout("timeout")` with `AsyncMock` to verify retry and Saga compensation paths!

What Topic Is Next → Next Topic 21.12: REST API Endpoint Testing Principles

21.12 REST API Endpoint Testing Principles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Verifying HTTP status code + response JSON schema + response headers + actual database state mutation + queued side effects on every endpoint test • Production Edge-Case & Pitfall Guard (REST API Endpoint Testing Principles) • Hands-On Python Implementation & Verification (REST API Endpoint Testing Principles) • REST API Endpoint Testing Principles — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular

architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for REST API Endpoint Testing Principles • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Asserting not just `resp.status_code == 201`, but also `assert db.get(Job, id) is not None` and `mock_celery.assert_called_once()`!

What Topic Is Next → Next Topic 21.13: Django Test Client

21.13 Django Test Client [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing Django views • Middleware headers • Redirects • Template context using `django.test.Client` • Production Edge-Case & Pitfall Guard (Django Test Client) • Hands-On Python Implementation & Verification (Django Test Client) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Django Test Client

Purpose: Verifying `/healthz`, security headers (`X-Content-Type-Options: nosniff`), and CSRF enforcement via `pytest-django`.

What Topic Is Next → Next Topic 21.14: DRF APIClient Testing

21.14 DRF APIClient Testing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `APIClient()` `format="json"` requests • Multipart `SimpleUploadedFile("resume.pdf" • B"%PDF-1.7..." • Content_type="application/pdf")` upload tests • Production Edge-Case & Pitfall Guard (DRF APIClient Testing) • Hands-On Python Implementation & Verification (DRF APIClient Testing) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for DRF APIClient Testing

Purpose: Testing PDF resume upload (`201`) vs fake `.exe` renamed `.pdf` (`400`) and locking N+1 query count with `django_assert_num_queries(2)`!

What Topic Is Next → Next Topic 21.15: FastAPI TestClient and AsyncClient Testing

21.15 FastAPI TestClient and AsyncClient Testing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Sync `TestClient(app)` vs native async `async with httpx.AsyncClient(transport=ASGITransport(app=app) • Base_url="http://test") as ac:` + `app.dependency_overrides` • Production Edge-Case & Pitfall Guard (FastAPI TestClient and AsyncClient Testing) • Hands-On Python Implementation & Verification (FastAPI TestClient and AsyncClient Testing) • FastAPI TestClient and AsyncClient Testing — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for FastAPI TestClient • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Testing `async def` FastAPI + `AsyncSession` routes natively on the same event loop via `httpx.AsyncClient(transport=ASGITransport(app=app))`!

What Topic Is Next → Next Topic 21.16: Isolated Test Databases and Transaction Rollbacks

21.16 Isolated Test Databases and Transaction Rollbacks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Nested SAVEPOINT rollback per test: create tables once per session • Open outer transaction + SAVEPOINT per test • `rollback()` in `< 3ms` after each test! • Production Edge-Case & Pitfall Guard (Isolated Test Databases and Transaction Rollbacks) • Hands-On Python Implementation & Verification (Isolated Test Databases and Transaction Rollbacks) • Isolated Test Databases and Transaction Rollbacks — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Isolated Test Databases

Purpose: Running 200 PostgreSQL integration tests in 6 seconds without `DROP TABLE` / `CREATE TABLE` overhead between tests.

What Topic Is Next → Next Topic 21.17: Multi-Component Integration Testing

21.17 Multi-Component Integration Testing [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing full vertical slices (`HTTP Router -> Pydantic -> Service -> SQLAlchemy -> PostgreSQL -> Redis Cache`) together with real containers in CI • Production Edge-Case & Pitfall Guard (Multi-Component Integration Testing) • Hands-On Python Implementation & Verification (Multi-Component Integration Testing) • Multi-Component Integration Testing — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multi-Component Integration Testing • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Verifying that `PATCH /products/{id}` commits to PostgreSQL AND invalidates Redis so the very next `GET` returns the updated price!

What Topic Is Next → Next Topic 21.18: Testing JWT-Protected Authentication Endpoints

21.18 Testing JWT-Protected Authentication Endpoints [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing valid Bearer JWT • Missing header (`401`) • Malformed token (`401`) • Expired `exp` token (`401`) • Production Edge-Case & Pitfall Guard (Testing JWT-Protected Authentication Endpoints) • Hands-On Python Implementation & Verification (Testing JWT-Protected Authentication Endpoints) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Testing JWT-Protected Authentication Endpoints

Purpose: Proving cryptographically tampered, expired, or blacklisted JWTs are rejected on every protected route.

What Topic Is Next → Next Topic 21.19: Testing Role-Based Authorization Boundaries

21.19 Testing Role-Based Authorization Boundaries [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Systematic RBAC & BOLA/IDOR matrix testing: Candidate calling Employer route (`403`) • Employer B editing Employer A's job (`403`/`404`) • Non-admin calling `/admin/*` (`403`) • Production Edge-Case & Pitfall Guard (Testing Role-Based Authorization Boundaries) • Hands-On Python Implementation & Verification (Testing Role-Based Authorization Boundaries) • Testing Role-Based Authorization Boundaries — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Testing Role-Based Authorization Boundaries

Purpose: Preventing horizontal and vertical privilege-escalation vulnerabilities before any PR can merge.

What Topic Is Next → Next Topic 21.20: Negative Testing for Invalid Payloads and Edge Cases

21.20 Negative Testing for Invalid Payloads and Edge Cases [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Testing missing required fields • Extra forbidden fields • Negative quantities • Empty strings • Production Edge-Case & Pitfall Guard (Negative Testing for Invalid Payloads and Edge Cases) • Hands-On Python Implementation & Verification (Negative Testing for Invalid Payloads and Edge Cases) • Idiomatic

PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Negative Testing for Invalid Payloads

Purpose: Guaranteeing malformed client inputs ALWAYS return clean `400`/`404`/`409`/`413`/`422` errors — NEVER an unhandled `500 Internal Server Error`!

What Topic Is Next → Next Topic 21.21: Measuring Test Coverage with Coverage.py

21.21 Measuring Test Coverage with Coverage.py [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `pytest --cov=app --cov-branch --cov-report=term-missing --cov-fail-under=80` • Branch coverage vs line coverage • Inspecting uncovered error branches • Production Edge-Case & Pitfall Guard (Measuring Test Coverage with Coverage.py) • Hands-On Python Implementation & Verification (Measuring Test Coverage with Coverage.py) • Measuring Test Coverage with Coverage.py — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Measuring Test Coverage with Coverage.py

Purpose: Using `--cov-report=term-missing` to spot untested rollback, compensation, and permission branches.

What Topic Is Next → Next Topic 21.22: Fast Python Linting with Ruff

21.22 Fast Python Linting with Ruff [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Configuring `Ruff` (10–100x faster Rust linter replacing Flake8 + isort + pyupgrade + bandit rules: `E` • `F` • `I` • `B` bugbear • Production Edge-Case & Pitfall Guard (Fast Python Linting with Ruff) • Hands-On Python Implementation & Verification (Fast Python Linting with Ruff) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Fast Python Linting with Ruff

Purpose: Catching mutable default arguments (`B006`), unused imports (`F401`), SQL/pickle security smells (`S`), and import sorting (`I`) in `50ms`!

What Topic Is Next → Next Topic 21.23: Deterministic Code Formatting with Black

21.23 Deterministic Code Formatting with Black [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `black --check .` / `ruff format --check .` zero-debate PEP 8 formatting (`line-length = 88/100`) eliminating style bikeshedding in PR diffs • Production Edge-Case & Pitfall Guard (Deterministic Code Formatting with Black) • Hands-On Python Implementation & Verification (Deterministic Code Formatting with Black) • Deterministic Code Formatting with Black — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Deterministic Code Formatting with Black • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Ensuring every developer's editor and CI format Python code identically.

What Topic Is Next → Next Topic 21.24: Static Type Checking with mypy Fundamentals

21.24 Static Type Checking with mypy Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running `mypy app` • `disallow_untyped_defs = true` • `check_untyped_defs = true` • Catching `None` attribute errors (`Optional` narrowing) before code ever runs! • Production Edge-Case & Pitfall Guard (Static Type Checking with mypy Fundamentals) • Hands-On Python Implementation & Verification (Static Type Checking with mypy Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Static Type Checking with mypy Fundamentals

Purpose: Catching `Item "None" of "User | None" has no attribute "email"` at static check time instead of `AttributeError` at 2 AM in production!

What Topic Is Next → Next Topic 21.25: Static Code Analysis in Pre-Commit Hooks

21.25 Static Code Analysis in Pre-Commit Hooks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `.pre-commit-config.yaml` (`ruff` • `black`/`ruff-format` • `mypy` • `detect-secrets` • Production Edge-Case & Pitfall Guard (Static Code Analysis in Pre-Commit Hooks) • Hands-On Python Implementation & Verification (Static Code Analysis in Pre-Commit Hooks) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Static Code Analysis in Pre-Commit Hooks

Purpose: Blocking accidental secret commits, merge-conflict markers, or lint errors locally BEFORE `git commit` even succeeds!

What Topic Is Next → Next Topic 21.26: Maintainability and Code Review Best Practices

21.26 Maintainability and Code Review Best Practices [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Small cohesive functions • Explicit typing • Cyclomatic complexity bounds • Structured error handling • Production Edge-Case & Pitfall Guard (Maintainability and Code Review Best Practices) • Hands-On Python Implementation & Verification (Maintainability and Code Review Best Practices) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Maintainability

Purpose: Reviewing teammates' Django and FastAPI PRs to senior industry engineering standards.

What Topic Is Next → Module 21 Capstone (Week 21 Lab Build: >= 80% Coverage Pytest, AsyncMock, Ruff & mypy Quality Gate) → Next Topic 22.1: Linux Operating System Fundamentals

Module 21 Hands-On Capstone Project (100 Marks): Week 21 Lab Build: >= 80% Coverage Pytest, AsyncMock, Ruff & mypy Quality Gate — Verify both DRF (`APIClient`) and FastAPI (`httpx.AsyncClient` + `ASGITransport`) services with transactional DB rollback fixtures, `@pytest.mark.parametrize`, `AsyncMock`, BOLA cross-tenant...

MODULE 22 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 30 TOPICS (240 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 22: Docker, Linux, Nginx and CI/CD

Containerize, automate, reverse-proxy, and deploy Python backends cleanly on Linux and cloud infrastructure: Linux OS & CLI (`pwd`/`ls`/`cd`/`mkdir`/`rm`/`cp`), `chmod`/`chown`, `ps`/`top`/`kill`, `curl`/`ping`/`ss`, env vars, SSH, server admin, Docker Engine architecture, images, containers, multi-stage non-root Python `Dockerfile`, volumes, networks, `docker-compose.yml` (App + PostgreSQL + Redis + Celery + Nginx), `.env`/secrets, Nginx reverse proxy, Uvicorn ASGI, Gunicorn + Uvicorn workers, GitHub Actions CI/CD pipelines, AWS (`EC2`, `S3`, `IAM`, `RDS`) fundamentals, and `/healthz` + log monitoring.

Objectives: ✓ Operate Ubuntu Linux servers confidently over SSH using filesystem, permission (`chmod`/`chown`), process (`ps`/`systemctl`), and network (`curl`/`ss`) commands. • ✓ Write lean, secure, non-root multi-stage `python:3.12-slim` `Dockerfile`s with `.dockerignore` and `PYTHONDONTWRITEBYTECODE=1` / `PYTHONUNBUFFERED=1`. • ✓ Orchestrate full-stack `docker-compose.yml` environments (FastAPI/Django + PostgreSQL + Redis + Celery Worker + Nginx) with healthchecks and persistent volumes. • ✓ Run Python in production behind Nginx reverse proxy using Gunicorn (`UvicornWorker` for FastAPI ASGI / WSGI workers for Django) with graceful shutdown. • ✓ Automate Lint -> Typecheck -> Pytest (with Postgres/Redis service containers) -> Docker Build -> Deploy in GitHub Actions and understand AWS EC2/S3/IAM/RDS.

22.1 Linux Operating System Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Kernel vs user space • FHS directory layout (`/etc` • `/var/log` • `/opt` • Production Edge-Case & Pitfall Guard (Linux Operating System Fundamentals) • Hands-On Python Implementation & Verification (Linux Operating System Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Operating System Fundamentals

Purpose: Navigating Ubuntu LTS servers and Debian-based `python:3.12-slim` Docker containers effortlessly.

What Topic Is Next → Next Topic 22.2: Essential Linux File Commands (pwd, ls, cd, mkdir, rm, cp)

22.2 Essential Linux File Commands (pwd, ls, cd, mkdir, rm, cp) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `pwd` • `ls -lah` • `cd` • `mkdir -p` • Production Edge-Case & Pitfall Guard (Essential Linux File Commands (pwd, ls, cd, mkdir, rm, cp)) • Hands-On Python Implementation & Verification (Essential Linux File Commands (pwd, ls, cd, mkdir, rm, cp)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Essential Linux File Commands (pwd, ls, cd, mkdir, r...

Purpose: Tailing live Gunicorn/Nginx logs (`tail -f /var/log/nginx/error.log`) and searching error timestamps with `grep`.

What Topic Is Next → Next Topic 22.3: Linux File Ownership and Permissions (chmod and chown)

22.3 Linux File Ownership and Permissions (chmod and chown) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Octal bits (`r=4 • W=2 • X=1`): `chmod 600` (`~/.ssh/id_ed25519` • `.env`) • Production Edge-Case & Pitfall Guard (Linux File Ownership and Permissions (chmod and chown)) • Hands-On Python Implementation & Verification (Linux File Ownership and Permissions (chmod and chown)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux File Ownership

Purpose: Locking `.env` secrets to `chmod 600` (owner read/write ONLY) and making `entrypoint.sh` executable (`chmod 755`).

What Topic Is Next → Next Topic 22.4: Process Management (ps, top, kill, systemctl)

22.4 Process Management (ps, top, kill, systemctl) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ps aux` • `top`/`htop` • Graceful `SIGTERM` (`kill -15`) vs last-resort `SIGKILL` (`kill -9`) • Exit codes (`0` vs non-zero) • Production Edge-Case & Pitfall Guard (Process Management (ps, top, kill, systemctl)) • Hands-On Python Implementation & Verification (Process Management (ps, top, kill, systemctl)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Process Management (ps, top, kill, systemctl)

Purpose: Sending `SIGTERM` during rolling deploys so Gunicorn/Uvicorn finishes in-flight HTTP requests before exiting cleanly!

What Topic Is Next → Next Topic 22.5: Networking Diagnostics (curl, ping, ss/netstat)

22.5 Networking Diagnostics (curl, ping, ss/netstat) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `curl -i -f http://localhost:8000/healthz` • Inspecting headers (`curl -I`) • Listening sockets `ss -tulpn` • `127.0.0.1` loopback vs `0.0.0.0` all interfaces • Production Edge-Case & Pitfall Guard (Networking Diagnostics (curl, ping, ss/netstat)) • Hands-On Python Implementation & Verification (Networking Diagnostics (curl, ping, ss/netstat)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Networking Diagnostics (curl, ping, ss/netstat)

Purpose: Diagnosing why a Docker container isn't reachable (`uvicorn --host 127.0.0.1` binds ONLY inside container loopback — containers MUST bind `0.0.0.0`!).

What Topic Is Next → Next Topic 22.6: Linux Environment Variables and Shell Profiles

22.6 Linux Environment Variables and Shell Profiles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `export KEY=VAL` • `env`/`printenv` • Subshell inheritance • `EnvironmentFile=` in `systemd` • Production Edge-Case & Pitfall Guard (Linux Environment Variables and Shell Profiles) • Hands-On Python Implementation & Verification (Linux Environment Variables and Shell Profiles) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Linux Environment Variables

Purpose: Passing credentials via environment variables/secret mounts rather than `--password=...` CLI flags.

What Topic Is Next → Next Topic 22.7: Secure Remote Server Access with SSH

22.7 Secure Remote Server Access with SSH [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ssh-keygen -t ed25519` • `~/.ssh/authorized_keys` • Disabling password auth & root login in `/etc/ssh/sshd_config` • `~/.ssh/config` • Production Edge-Case & Pitfall Guard (Secure Remote Server Access with SSH) • Hands-On Python Implementation & Verification (Secure Remote Server Access with SSH) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Secure Remote Server Access with SSH

Purpose: Connecting to AWS EC2 Ubuntu servers securely using Ed25519 SSH keys with password login disabled.

What Topic Is Next → Next Topic 22.8: Basic Linux Server Administration

22.8 Basic Linux Server Administration [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ufw` firewall (`allow OpenSSH` • `allow 80 • 443/tcp` • Keep `5432`/`6379` private!) • Production Edge-Case & Pitfall Guard (Basic Linux Server Administration) • Hands-On Python Implementation & Verification (Basic Linux Server Administration) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Basic Linux Server Administration

Purpose: Ensuring PostgreSQL (`5432`) and Redis (`6379`) are NEVER exposed to `0.0.0.0` on the public internet!

What Topic Is Next → Next Topic 22.9: Containers vs Virtual Machines (Docker Introduction)

22.9 Containers vs Virtual Machines (Docker Introduction) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Shared Linux kernel namespaces (`pid` • `net` • `mnt` • `user`) + `cgroups` resource limits vs full guest-OS hypervisor VMs • Production Edge-Case & Pitfall Guard (Containers vs Virtual Machines (Docker Introduction)) • Hands-On Python Implementation & Verification (Containers vs Virtual Machines (Docker Introduction)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Containers vs Virtual Machines (Docker Introduction)

Purpose: Shipping the exact Python 3.12.x, `libpq`, and pinned wheel dependencies from laptop -> GitHub Actions -> production.

What Topic Is Next → Next Topic 22.10: Docker Engine Setup

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

Subtopics Covered: Docker Desktop (WSL 2 backend on Windows) & Docker Engine CE on Ubuntu Linux • Verifying `docker info` • BuildKit • Production Edge-Case & Pitfall Guard (Docker Engine Setup) • Hands-On Python Implementation & Verification (Docker Engine Setup) • Docker Engine Setup — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Engine Setup

Purpose: Building Linux containers locally on Windows WSL 2 and deploying on Linux servers.

What Topic Is Next → Next Topic 22.11: Docker Architecture (Client, Daemon, Registry)

22.11 Docker Architecture (Client, Daemon, Registry) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `docker` CLI -> REST socket -> `dockerd`/`containerd` -> OCI runtime (`runc`) • UnionFS copy-on-write layers • Registries (GHCR `ghcr.io` • ECR • Production Edge-Case & Pitfall Guard (Docker Architecture (Client, Daemon, Registry)) • Hands-On Python Implementation & Verification (Docker Architecture (Client, Daemon, Registry)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker Architecture (Client, Daemon, Registry)

Purpose: Understanding why layer order in `Dockerfile` determines whether rebuilds take `2s` (cached wheels) or `90s`!

What Topic Is Next → Next Topic 22.12: Pulling and Tagging Docker Images

22.12 Pulling and Tagging Docker Images [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Choosing `python:3.12-slim-bookworm` over bloated `python:3.12` (`~1 GB` vs `~130 MB`) or musl `alpine` (breaks C-extension manylinux wheels!) • Tagging `:git-sha` + `:v1.0.0` • Production Edge-Case & Pitfall Guard (Pulling and Tagging Docker Images) • Hands-On Python Implementation & Verification (Pulling and Tagging Docker Images) • Pulling and Tagging Docker Images — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Pulling • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Using Debian-based `python:3.12-slim` so precompiled `psycopg`/`pydantic-core`/`cryptography` binary wheels install in 3 seconds!

What Topic Is Next → Next Topic 22.13: Running and Managing Containers

22.13 Running and Managing Containers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Ephemeral stateless containers • PID 1 signal forwarding (`exec` form `CMD ["gunicorn" • ...]` • NEVER shell string form!) • Production Edge-Case & Pitfall Guard (Running and Managing Containers) • Hands-On Python Implementation & Verification (Running and Managing Containers) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Running

Purpose: Using JSON exec-array `CMD ["uvicorn", "app.main:app", ...]` so `uvicorn` runs as PID 1 and receives `SIGTERM` directly on `docker stop`!

What Topic Is Next → Next Topic 22.14: Writing Multi-Stage Python Dockerfiles

22.14 Writing Multi-Stage Python Dockerfiles [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Stage 1 (`builder` compiling wheels into `/opt/venv`) -> Stage 2 (`runtime` copying `/opt/venv` + `app/`) • `.dockerignore` • `PYTHONDONTWRITEBYTECODE=1` • `PYTHONUNBUFFERED=1` • Production Edge-Case & Pitfall Guard (Writing Multi-Stage Python Dockerfiles) • Hands-On Python Implementation & Verification (Writing Multi-Stage Python Dockerfiles) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Writing Multi-Stage Python Dockerfiles

Purpose: Shipping a hardened `160 MB` production image containing zero compilers (`gcc`), zero `.git` history, unbuffered stdout logs, and non-root execution.

What Topic Is Next → Next Topic 22.15: Docker CLI Lifecycle Commands (build, run, stop, rm, logs)

22.15 Docker CLI Lifecycle Commands (build, run, stop, rm, logs) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `docker build -t ... .` • `docker run --rm -e ... -p ...` • `docker ps -a` • `docker logs -f --tail 100` • Production Edge-Case & Pitfall Guard (Docker CLI Lifecycle Commands (build, run, stop, rm, logs)) • Hands-On Python Implementation & Verification (Docker CLI Lifecycle Commands (build, run, stop, rm, logs)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Docker CLI Lifecycle Commands (build, run, stop, rm,...

Purpose: Debugging container startup failures via `docker logs` and running one-off admin commands via `docker compose exec api python manage.py ...`.

What Topic Is Next → Next Topic 22.16: Persistent Storage with Docker Volumes

22.16 Persistent Storage with Docker Volumes [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Named Docker volumes (`postgres_data:/var/lib/postgresql/data` • `redis_data:/data` • `media_uploads`) surviving `docker compose down` vs `down -v` (DESTROYS volumes!) • Production Edge-Case & Pitfall Guard (Persistent Storage with Docker Volumes) • Hands-On Python Implementation & Verification (Persistent Storage with Docker Volumes) • Persistent Storage with Docker Volumes — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Persistent Storage with Docker Volumes

Purpose: Persisting PostgreSQL database files across application container upgrades and restarts.

What Topic Is Next → Next Topic 22.17: User-Defined Container Networks

22.17 User-Defined Container Networks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Custom bridge networks (`frontend_net` for `nginx <-> api` • `backend_net` internal for `api <-> db/redis`) • Automatic Docker DNS service-name resolution • Production Edge-Case & Pitfall Guard (User-Defined Container Networks) • Hands-On Python Implementation & Verification (User-Defined Container Networks) • User-Defined Container Networks — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for User-Defined Container Networks

Purpose: Isolating `postgres` and `redis` on `backend_net` so `nginx` cannot even reach `postgres` directly at the network layer.

What Topic Is Next → Next Topic 22.18: Multi-Container Stacks with Docker Compose

22.18 Multi-Container Stacks with Docker Compose [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Authoring `compose.yaml` (`api` • `worker` • `beat` • `db` • Production Edge-Case & Pitfall Guard (Multi-Container Stacks with Docker Compose) • Hands-On Python Implementation & Verification (Multi-Container Stacks with Docker Compose) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Multi-Container Stacks with Docker Compose

Purpose: Booting the entire 6-container Job Portal stack in one deterministic `docker compose up -d --build` command!

What Topic Is Next → Next Topic 22.19: .env Configuration and Secrets Handling

22.19 .env Configuration and Secrets Handling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Docker Compose variable substitution `${POSTGRES_PASSWORD:?required}` • `env_file` • Docker/GitHub Secrets • Never baking credentials into image `ENV` or `ARG` layers • Production Edge-Case & Pitfall Guard (.env Configuration and Secrets Handling) • Hands-On Python Implementation & Verification (.env Configuration and Secrets Handling) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for .env Configuration

Purpose: Using `${JWT_SECRET:?Set JWT_SECRET in .env}` in `compose.yaml` so missing secrets fail immediately with a clear message.

What Topic Is Next → Next Topic 22.20: Containerizing PostgreSQL with Healthchecks

22.20 Containerizing PostgreSQL with Healthchecks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `postgres:16-alpine` • `pg_isready -U $POSTGRES_USER -d $POSTGRES_DB` healthcheck • `postgres_data` named volume • Running migrations as a dedicated init/release step • Production Edge-Case & Pitfall Guard (Containerizing PostgreSQL with Healthchecks) • Hands-On Python Implementation & Verification (Containerizing PostgreSQL with Healthchecks) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Containerizing PostgreSQL with Healthchecks

Purpose: Running `alembic upgrade head` (or `manage.py migrate --noinput`) ONCE in a `migrate` service before `api` replicas start (avoiding 4 replicas racing `migrate` simultaneo...

What Topic Is Next → Next Topic 22.21: Containerizing Redis for Cache and Queues

22.21 Containerizing Redis for Cache and Queues [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `redis:7-alpine` • `redis-cli ping` healthcheck • `--appendonly yes` AOF durability for Celery/rate-limit state • `--maxmemory` bounds • Production Edge-Case & Pitfall Guard (Containerizing Redis for Cache and Queues) • Hands-On Python Implementation & Verification (Containerizing Redis for Cache and Queues) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Containerizing Redis for Cache

Purpose: Running `redis:7-alpine` with `--maxmemory 256mb` + AOF persistence backing Celery and cache.

What Topic Is Next → Next Topic 22.22: Nginx Reverse Proxy Fundamentals

22.22 Nginx Reverse Proxy Fundamentals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Why Nginx sits in front of Gunicorn/Uvicorn: TLS 1.3 termination • Slow-client buffering (`client_max_body_size 6M`) • Gzip/zstd compression • Static files • Production Edge-Case & Pitfall Guard (Nginx Reverse Proxy Fundamentals) • Hands-On Python Implementation & Verification (Nginx Reverse Proxy Fundamentals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Nginx Reverse Proxy Fundamentals

Purpose: Rejecting oversized `> 6 MB` uploads at Nginx edge before bytes ever hit Python workers and forwarding `X-Forwarded-Proto https`.

What Topic Is Next → Next Topic 22.23: Uvicorn ASGI Production Server

22.23 Uvicorn ASGI Production Server [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `uvicorn` event loop (`uvloop`) • HTTP parser (`httptools`) • `--proxy-headers` • `--timeout-keep-alive` • Production Edge-Case & Pitfall Guard (Uvicorn ASGI Production Server) • Hands-On Python Implementation & Verification (Uvicorn ASGI Production Server) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Uvicorn ASGI Production Server

Purpose: Serving async FastAPI endpoints at multi-thousand RPS per CPU core.

What Topic Is Next → Next Topic 22.24: Gunicorn Process Manager with Uvicorn Workers

22.24 Gunicorn Process Manager with Uvicorn Workers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Running `gunicorn app.main:app -k uvicorn_worker.UvicornWorker --workers 2 --graceful-timeout 30 --max-requests 2000 --max-requests-jitter 200` (process supervision + multi-core scaling!) • Production Edge-Case & Pitfall Guard (Gunicorn Process Manager with Uvicorn Workers) • Hands-On Python Implementation & Verification (Gunicorn Process Manager with Uvicorn Workers) • Gunicorn Process Manager with Uvicorn Workers — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Gunicorn Process Manager with Uvicorn Workers • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Combining Gunicorn’s battle-tested multi-process master supervisor (auto-restarting crashed workers & preventing memory leaks via `--max-requests`) with Uvicorn’s async A...

What Topic Is Next → Next Topic 22.25: End-to-End Python Backend Deployment

22.25 End-to-End Python Backend Deployment [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Zero-surprise release sequence: Pull image -> Run DB migrations (backward-compatible!) -> Healthcheck new container -> Swap traffic -> Verify `/readyz` & logs • Production Edge-Case & Pitfall Guard (End-to-End Python Backend Deployment) • Hands-On Python Implementation & Verification (End-to-End Python Backend Deployment) • End-to-End Python Backend Deployment — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for End-to-End Python Backend Deployment • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Deploying every major project cleanly so interviewers can hit live Swagger `/docs` URLs.

What Topic Is Next → Next Topic 22.26: CI/CD Concepts (Continuous Integration and Delivery)

22.26 CI/CD Concepts (Continuous Integration and Delivery) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Automated quality gates on every PR (CI) vs automated staging/production artifact promotion on `main` merge/tag (CD) • Production Edge-Case & Pitfall Guard (CI/CD Concepts (Continuous Integration and Delivery)) • Hands-On Python Implementation & Verification (CI/CD Concepts (Continuous Integration and Delivery)) • CI/CD Concepts (Continuous Integration and Delivery) — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for CI/CD Concepts (Continuous Integration and Delivery) • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Guaranteeing broken tests, `Ruff` lint errors, or missing migrations physically cannot merge into `main`.

What Topic Is Next → Next Topic 22.27: GitHub Actions Workflows (.github/workflows)

22.27 GitHub Actions Workflows (.github/workflows) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Triggers (`on: [push • Pull_request]`) • Ubuntu runners • `actions/setup-python@v5` (`cache: "pip"`) • Production Edge-Case & Pitfall Guard (GitHub Actions Workflows (.github/workflows)) • Hands-On Python Implementation & Verification (GitHub Actions Workflows (.github/workflows)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for GitHub Actions Workflows (.github/workflows)

Purpose: Spinning up real `postgres:16-alpine` and `redis:7-alpine` `services:` inside GitHub Actions for every Pull Request!

What Topic Is Next → Next Topic 22.28: Automated Lint, Pytest, Build and Deploy Pipeline

22.28 Automated Lint, Pytest, Build and Deploy Pipeline [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Chaining `lint` (`ruff check` • `ruff format --check` • `mypy`) + `test` (`alembic upgrade head` • `pytest --cov-fail-under=80`) + `docker` (`docker/build-push-action`) -> SSH/Webhook deploy only when tests pass (`needs: [lint • Production Edge-Case & Pitfall Guard (Automated Lint, Pytest, Build and Deploy Pipeline) • Hands-On Python Implementation & Verification (Automated Lint, Pytest, Build and Deploy Pipeline) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Automated Lint

Purpose: Ensuring `deploy` job has `needs: [lint, test]` so a broken test halts deployment 100% of the time!

What Topic Is Next → Next Topic 22.29: AWS Cloud Fundamentals (EC2, S3, IAM, RDS)

22.29 AWS Cloud Fundamentals (EC2, S3, IAM, RDS) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Practical backend AWS literacy: `IAM` least-privilege roles (no root keys!) • `EC2` + Security Groups • `S3` private buckets + presigned resume URLs (`boto3`) • Managed `RDS` PostgreSQL automated backups • Production Edge-Case & Pitfall Guard (AWS Cloud Fundamentals (EC2, S3, IAM, RDS)) • Hands-On Python Implementation & Verification (AWS Cloud Fundamentals (EC2, S3, IAM, RDS)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for AWS Cloud Fundamentals (EC2, S3, IAM, RDS)

Purpose: Storing candidate PDF resumes in a PRIVATE `S3` bucket and generating 5-minute presigned download URLs via `boto3` for authorized employers!

What Topic Is Next → Next Topic 22.30: Application Logs, Readiness and /health Checks

22.30 Application Logs, Readiness and /health Checks [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Separating Liveness `/healthz` (is Python process alive? `200`) from Readiness `/readyz` (can it query PostgreSQL `SELECT 1` & `PING` Redis? `200` vs `503`) • Plus Sentry error tracking basics • Production Edge-Case & Pitfall Guard (Application Logs, Readiness and /health Checks) • Hands-On Python Implementation & Verification (Application Logs, Readiness and /health Checks) • Application Logs, Readiness and /health Checks — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Application Logs • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards

Purpose: Returning `503 Service Unavailable` on `/readyz` when DB pool fails while keeping `/healthz` fast so load balancers stop routing traffic gracefully.

What Topic Is Next → Module 22 Capstone (Week 22 Lab Build: Non-Root Multi-Stage Docker Compose, Nginx & GitHub Actions CI) → Next Topic 23.1: Interview Review: Core Python Types, Mutability and Operators

Module 22 Hands-On Capstone Project (100 Marks): Week 22 Lab Build: Non-Root Multi-Stage Docker Compose, Nginx & GitHub Actions CI — Package a FastAPI/Django + PostgreSQL 16 + Redis 7 + Celery Worker + Nginx reverse-proxy stack using a multi-stage non-root `python:3.12-slim` `Dockerfile`, health-checked `compose.yaml`, Gu...

MODULE 23 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 24 TOPICS (192 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 23: Python Coding Challenges and Backend Technical Interviews

Consolidate all 6 months into crisp technical-interview and timed machine-coding mastery: complete 24-area revision across Python internals, OOP, decorators/generators, GIL/concurrency/asyncio, DSA, SQL joins/window functions/indexing/ACID, Django/DRF, FastAPI/Pydantic, REST/JWT/OAuth2, Microservices, Redis/Celery/Kafka, Pytest, Docker/Git/Deployment, System Design, PLUS all 12 Backend Machine-Coding Challenges (Auth Module, Paginated API, Rate Limiter, Redis Cache, Celery Mailer, Async API, File Upload, Order Workflow, URL Shortener) and 100–150 Python DSA completion.

Objectives: ✓ Answer deep "how it works under the hood" technical viva questions across Python 3.12, SQL/PostgreSQL, Django/DRF, FastAPI, Redis, Celery, Kafka, and Docker. • ✓ Build clean, tested backend modules from scratch under 60–90 minute machine-coding constraints (URL Shortener, Rate Limiter,

Reusable JWT/RBAC, Order Workflow). • ✓ Solve timed Easy & Medium Python DSA problems (`100–150` program target) stating exact Time & Space Complexity before and after coding. • ✓ Walk through Junior/Mid-level Backend System Design diagrams (URL Shortener, Notification Pipeline, Job Portal, E-Commerce Checkout) clearly. • ✓ Present your own GitHub commits, SQL queries, migrations, and architecture decisions confidently without memorized buzzwords.

23.1 Interview Review: Core Python Types, Mutability and Operators [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `is` vs `==` • Small-int interning trap • Mutable vs immutable types • Pass-by-object-reference • Production Edge-Case & Pitfall Guard (Interview Review: Core Python Types, Mutability and Operators) • Hands-On Python Implementation & Verification (Interview Review: Core Python Types, Mutability and Operators) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Explaining crisp memory-reference diagrams on tricky Python output-prediction questions.

What Topic Is Next → Next Topic 23.2: Interview Review: Lists, Tuples, Sets and Dictionary Internals

23.2 Interview Review: Lists, Tuples, Sets and Dictionary Internals [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Over-allocated `list` array • `dict` open-addressing compact hash table (`O(1)` avg) • Hashability requirements (`__hash__` + `__eq__`) • `copy` vs `deepcopy` • Production Edge-Case & Pitfall Guard (Interview Review: Lists, Tuples, Sets and Dictionary Internals) • Hands-On Python Implementation & Verification (Interview Review: Lists, Tuples, Sets and Dictionary Internals) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Explaining why Python 3.7+ `dict` preserves insertion order while using compact index + entry arrays.

What Topic Is Next → Next Topic 23.3: Interview Review: Functions, *args/kwargs, LEGB Scope and Recursion**

23.3 Interview Review: Functions, *args/kwargs, LEGB Scope and Recursion** [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Mutable default argument trap • Late-binding closure trap • LEGB + `nonlocal` • Keyword-only `*` args • Production Edge-Case & Pitfall Guard (Interview Review: Functions, *args/**kwargs, LEGB Scope and Recursion) • Hands-On Python Implementation & Verification (Interview Review: Functions, *args/**kwargs, LEGB Scope and Recursion) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Spotting and fixing mutable-default and late-binding closure bugs in live code-review rounds.

What Topic Is Next → Next Topic 23.4: Interview Review: Python OOP, MRO, Duck Typing and ABCs

23.4 Interview Review: Python OOP, MRO, Duck Typing and ABCs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `__new__` vs `__init__` • `@classmethod` vs `@staticmethod` • `@property` • C3 MRO + cooperative `super()` • Production Edge-Case & Pitfall Guard (Interview Review: Python OOP, MRO, Duck Typing and ABCs) • Hands-On Python Implementation & Verification (Interview Review: Python OOP, MRO, Duck Typing and ABCs) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Articulating composition vs inheritance and `ABC` nominal vs `Protocol` structural typing clearly.

What Topic Is Next → Next Topic 23.5: Interview Review: Decorators, Generators, Iterators and Context Managers

23.5 Interview Review: Decorators, Generators, Iterators and Context Managers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Parameterized `@decorator(arg)` 3-level function structure • `@wraps` • `Iterator` vs `Iterable` • `yield` memory savings • Production Edge-Case & Pitfall Guard (Interview Review: Decorators, Generators, Iterators and Context Managers) • Hands-On Python Implementation & Verification (Interview Review: Decorators, Generators, Iterators and Context Managers) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Writing a parameterized `@retry(times=3, exceptions=(TimeoutError,))` decorator live on a whiteboard/shared editor in 5 minutes!

What Topic Is Next → Next Topic 23.6: Interview Review: CPython Memory, Reference Counting, GC and GIL

23.6 Interview Review: CPython Memory, Reference Counting, GC and GIL [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `ob_refcnt` + generational cyclic `gc` • `__slots__` memory optimization • What the GIL locks • Why C extensions / OS I/O syscalls release the GIL • Production Edge-Case & Pitfall Guard (Interview Review: CPython Memory, Reference Counting, GC and GIL) • Hands-On Python Implementation & Verification (Interview Review: CPython Memory, Reference Counting, GC and GIL) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Giving an accurate, nuanced engineering answer to "How does memory management and the GIL work in CPython?".

What Topic Is Next → Next Topic 23.7: Interview Review: Threading vs Multiprocessing vs Asyncio

23.7 Interview Review: Threading vs Multiprocessing vs Asyncio [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Exact comparison across memory sharing • OS context switching • CPU-bound vs blocking I/O vs high-concurrency non-blocking I/O • `asyncio.to_thread` • Production Edge-Case & Pitfall Guard (Interview Review: Threading vs Multiprocessing vs Asyncio) • Hands-On Python Implementation & Verification (Interview Review: Threading vs Multiprocessing vs Asyncio) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Selecting and justifying concurrency architecture for web scrapers, image resize pipelines, and WebSocket servers.

What Topic Is Next → Next Topic 23.8: Interview Review: Python DSA Patterns (Arrays, Strings, Trees, Graphs)

23.8 Interview Review: Python DSA Patterns (Arrays, Strings, Trees, Graphs) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Pattern recognition: sorted array -> Two Pointers / Binary Search • Contiguous subarray -> Sliding Window / Prefix Sum + Dict • Top-K -> `heapq` • Shortest unweighted path -> `deque` BFS • Production Edge-Case & Pitfall Guard (Interview Review: Python DSA Patterns (Arrays, Strings, Trees, Graphs)) • Hands-On Python Implementation & Verification (Interview Review: Python DSA Patterns (Arrays, Strings, Trees, Graphs)) • Idiomatic PBCP Implementation: clean

modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Identifying the optimal `O(n)` / `O(n log n)` pattern within the first 3 minutes of a coding round.

What Topic Is Next → Next Topic 23.9: Interview Review: Complex SQL Joins, Subqueries, CTEs and Window Functions

23.9 Interview Review: Complex SQL Joins, Subqueries, CTEs and Window Functions [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Top-N per department (`DENSE_RANK`) • Running totals • YoY/MoM `LAG` • `LEFT JOIN` anti-joins • Production Edge-Case & Pitfall Guard (Interview Review: Complex SQL Joins, Subqueries, CTEs and Window Functions) • Hands-On Python Implementation & Verification (Interview Review: Complex SQL Joins, Subqueries, CTEs and Window Functions) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Writing bug-free analytical SQL live on screen during backend SQL interview rounds.

What Topic Is Next → Next Topic 23.10: Interview Review: PostgreSQL Indexing, ACID Isolation and Normalization

23.10 Interview Review: PostgreSQL Indexing, ACID Isolation and Normalization [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Composite B-Tree leftmost prefix • `EXPLAIN ANALYZE` • MVCC • `READ COMMITTED` vs `REPEATABLE READ` • Production Edge-Case & Pitfall Guard (Interview Review: PostgreSQL Indexing, ACID Isolation and Normalization) • Hands-On Python Implementation & Verification (Interview Review: PostgreSQL Indexing, ACID Isolation and Normalization) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Diagnosing slow production queries and race-condition double-bookings in scenario interviews.

What Topic Is Next → Next Topic 23.11: Interview Review: Django ORM, Middleware, Signals and N+1 Tuning

23.11 Interview Review: Django ORM, Middleware, Signals and N+1 Tuning [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Lazy QuerySets • `select_related` vs `prefetch_related` • `F()` / `Q()` / `annotate()` • `transaction.atomic()` + `on_commit()` • Production Edge-Case & Pitfall Guard (Interview Review: Django ORM, Middleware, Signals and N+1 Tuning) • Hands-On Python Implementation & Verification (Interview Review: Django ORM, Middleware, Signals and N+1 Tuning) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Answering Django-specific senior backend questions with real production depth.

What Topic Is Next → Next Topic 23.12: Interview Review: DRF Serializers, ViewSets, Permissions and Throttling

23.12 Interview Review: DRF Serializers, ViewSets, Permissions and Throttling [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `Serializer` vs `ModelSerializer` • `validate_<field>` vs `validate` • `APIView` vs `ModelViewSet` • `has_permission` vs `has_object_permission` • Production Edge-Case & Pitfall Guard (Interview Review: DRF Serializers, ViewSets, Permissions and Throttling) • Hands-On Python Implementation & Verification (Interview Review: DRF Serializers, ViewSets, Permissions and Throttling) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Explaining exact DRF request validation, permission checking, and N+1 prevention.

What Topic Is Next → Next Topic 23.13: Interview Review: FastAPI Depends(), Pydantic v2 and Async SQLAlchemy

23.13 Interview Review: FastAPI Depends(), Pydantic v2 and Async SQLAlchemy [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `async def` vs `def` event-loop behavior • Pydantic v2 `ConfigDict`/`response_model` • `Depends` caching & `yield` teardown • `AsyncSession(expire_on_commit=False)` • Production Edge-Case & Pitfall Guard (Interview Review: FastAPI Depends(), Pydantic v2 and Async SQLAlchemy) • Hands-On Python Implementation & Verification (Interview Review: FastAPI Depends(), Pydantic v2 and Async SQLAlchemy) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Explaining why FastAPI excels at async I/O and how `Depends` powers both auth and clean testing.

What Topic Is Next → Next Topic 23.14: Interview Review: REST API Design, Idempotency and HTTP Status Semantics

23.14 Interview Review: REST API Design, Idempotency and HTTP Status Semantics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Resource naming • `PUT` vs `PATCH` • `401` vs `403` • `400` vs `422` • Production Edge-Case & Pitfall Guard (Interview Review: REST API Design, Idempotency and HTTP Status Semantics) • Hands-On Python Implementation & Verification (Interview Review: REST API Design, Idempotency and HTTP Status Semantics) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Designing clean API contracts live during "Design the APIs for X" interview rounds.

What Topic Is Next → Next Topic 23.15: Interview Review: Stateless JWT, Refresh Rotation and OAuth2 Flows

23.15 Interview Review: Stateless JWT, Refresh Rotation and OAuth2 Flows [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Access + Refresh token lifecycle • Token rotation & reuse detection • `HttpOnly` cookie vs memory storage • RBAC vs BOLA/IDOR • Production Edge-Case & Pitfall Guard (Interview Review: Stateless JWT, Refresh Rotation and OAuth2 Flows) • Hands-On Python Implementation & Verification (Interview Review: Stateless JWT, Refresh Rotation and OAuth2 Flows) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Defending authentication & security trade-offs clearly in backend security interviews.

What Topic Is Next → Next Topic 23.16: Interview Review: Microservices Boundaries, Saga and Circuit Breakers

23.16 Interview Review: Microservices Boundaries, Saga and Circuit Breakers [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Database-per-service • Dual-Write & Transactional Outbox • Choreography vs Orchestration Saga • Circuit Breaker states • Production Edge-Case & Pitfall Guard (Interview Review: Microservices Boundaries, Saga and Circuit Breakers) • Hands-On Python Implementation & Verification (Interview Review: Microservices Boundaries, Saga and Circuit Breakers) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Walking through failure recovery when Payment succeeds, Inventory fails, or RabbitMQ/Kafka redelivers an event.

What Topic Is Next → Next Topic 23.17: Machine-Coding Lab: Reusable JWT Auth & RBAC Middleware Module

23.17 Machine-Coding Lab: Reusable JWT Auth & RBAC Middleware Module [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Machine-coding challenges #1 • #2 & #5: Python Employee Management App + Reusable JWT Authentication Module + Role-Based Authorization (`Admin` • `Manager` • `Member`) • Production Edge-Case & Pitfall Guard (Machine-Coding Lab: Reusable JWT Auth & RBAC Middleware Module) • Hands-On Python Implementation & Verification (Machine-Coding Lab: Reusable JWT Auth & RBAC Middleware Module) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine-Coding Lab

Purpose: Building a clean, drop-in FastAPI/DRF Auth + RBAC package with registration, Argon2 hashing, JWT rotation, and `pytest` tests in 60 minutes.

What Topic Is Next → Next Topic 23.18: Machine-Coding Lab: Cursor/Offset Paginated & Filtered Search API

23.18 Machine-Coding Lab: Cursor/Offset Paginated & Filtered Search API [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Machine-coding challenges #3 & #4: Complete CRUD REST API + Dynamic Multi-Filter • Full-Text Search • Whitelisted Sorting & Bounded Pagination API • Production Edge-Case & Pitfall Guard (Machine-Coding Lab: Cursor/Offset Paginated & Filtered Search API) • Hands-On Python Implementation & Verification (Machine-Coding Lab: Cursor/Offset Paginated & Filtered Search API) • Machine-Coding Lab: Cursor/Offset Paginated & Filtered Search API — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine-Coding Lab

Purpose: Implementing a fast, indexed Product/Job Search endpoint handling optional filters (`0`/`False`-safe!) and pagination cleanly.

What Topic Is Next → Next Topic 23.19: Machine-Coding Lab: Redis Cache-Aside & Sliding-Window API Rate Limiter

23.19 Machine-Coding Lab: Redis Cache-Aside & Sliding-Window API Rate Limiter [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Machine-coding challenges #7 & #10: Redis Cache-Aside wrapper with TTL/invalidation + Sliding-Window / Token-Bucket API Rate Limiter returning `429` & `Retry-After` • Production Edge-Case & Pitfall Guard (Machine-Coding Lab: Redis Cache-Aside & Sliding-Window API Rate Limiter) • Hands-On Python Implementation & Verification (Machine-Coding Lab: Redis Cache-Aside & Sliding-Window API Rate Limiter) • Machine-Coding Lab: Redis Cache-Aside & Sliding-Window API Rate Limiter — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine-Coding Lab • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Coding a working Redis sliding-window rate limiter (`X-RateLimit-Remaining`, `Retry-After`) — one of the most popular backend machine-coding rounds!

What Topic Is Next → Next Topic 23.20: Machine-Coding Lab: Celery + Redis Background Email & Order Workflow

23.20 Machine-Coding Lab: Celery + Redis Background Email & Order Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Machine-coding challenges #6 & #11: Background Email Notification Service (retries + idempotency) + Transactional Order Processing State Machine (`PENDING -> PAID -> SHIPPED / CANCELLED`) • Production Edge-Case & Pitfall Guard (Machine-Coding Lab: Celery + Redis Background Email & Order Workflow) • Hands-On Python Implementation & Verification (Machine-Coding Lab: Celery + Redis Background Email & Order Workflow) • Machine-Coding Lab: Celery + Redis Background Email & Order Workflow — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine-Coding Lab • Production Hardening & Security: input validation, least-privilege access & OWASP safeguards • Diagnostics, Profiling & Telemetry: structured telemetry, profiler traces & CI checks

Purpose: Enforcing valid order state transitions atomically with `SELECT ... FOR UPDATE` and queuing post-commit Celery email notifications.

What Topic Is Next → Next Topic 23.21: Machine-Coding Lab: Validated Resume/File Upload & Async Aggregator API

23.21 Machine-Coding Lab: Validated Resume/File Upload & Async Aggregator API [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Machine-coding challenges #8 & #9: Non-Blocking Async Aggregator API (`asyncio.gather` + `httpx` timeouts) + Secure Multipart File Upload Service (UUID rename • Magic bytes • Size cap) • Production Edge-Case & Pitfall Guard (Machine-Coding Lab: Validated Resume/File Upload & Async Aggregator API) • Hands-On Python Implementation & Verification (Machine-Coding Lab: Validated Resume/File Upload & Async Aggregator API) • Machine-Coding Lab: Validated Resume/File Upload & Async Aggregator API — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine-Coding Lab

Purpose: Aggregating 3 upstream JSON feeds concurrently in `< 150ms` and validating binary file uploads safely.

What Topic Is Next → Next Topic 23.22: Machine-Coding Lab: URL Shortener Backend with Analytics & TTL

23.22 Machine-Coding Lab: URL Shortener Backend with Analytics & TTL [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Machine-coding challenge #12: Designing & coding a URL Shortener (`POST /shorten` Base62 short code • `GET /{code}` `302` redirect cached in Redis • Click counter • Optional TTL expiry) • Production Edge-Case & Pitfall Guard (Machine-Coding Lab: URL Shortener Backend with Analytics & TTL) • Hands-On Python Implementation & Verification (Machine-Coding Lab: URL Shortener Backend with Analytics & TTL) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Machine-Coding Lab

Purpose: Solving the #1 Junior/Mid Backend System Design + Machine-Coding challenge end-to-end in FastAPI + PostgreSQL + Redis!

What Topic Is Next → Next Topic 23.23: Interview Review: Pytest Fixtures, Mocking, Docker & Git PR Troubleshooting

23.23 Interview Review: Pytest Fixtures, Mocking, Docker & Git PR Troubleshooting

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

Subtopics Covered: `conftest.py` fixtures • `patch` lookup rule • `AsyncMock` • Multi-stage non-root `Dockerfile` • Production Edge-Case & Pitfall Guard (Interview Review: Pytest Fixtures, Mocking, Docker & Git PR Troubleshooting) • Hands-On Python Implementation & Verification (Interview Review: Pytest Fixtures, Mocking, Docker & Git PR Troubleshooting) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Interview Review

Purpose: Answering practical engineering workflow questions ("How do you roll back a bad migration/commit on `main`?", "Why did your Docker container exit with 137 OOM?").

What Topic Is Next → Next Topic 23.24: Backend System Design Walkthroughs for Junior/Mid Python Roles

23.24 Backend System Design Walkthroughs for Junior/Mid Python Roles

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

Subtopics Covered: Structured 5-step System Design method: 1) Functional & Non-Functional Requirements • 2) Back-of-Envelope Traffic/Storage • 3) API Contracts • 4) PostgreSQL Schema & Indexes • Production Edge-Case & Pitfall Guard (Backend System Design Walkthroughs for Junior/Mid Python Roles) • Hands-On Python Implementation & Verification (Backend System Design Walkthroughs for Junior/Mid Python Roles) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Backend System Design Walkthroughs for Junior/Mid Py...

Purpose: Delivering structured, grounded 20-minute architecture designs without jumping prematurely to exotic planet-scale buzzwords.

What Topic Is Next → Module 23 Capstone (Week 23 Lab Build: URL Shortener + Sliding-Window Rate Limiter + 120-DSA Portfolio) → Next Topic 24.1: Capstone Module 1: Authentication Service (Registration, Login, JWT & Refresh Rotation)

Module 23 Hands-On Capstone Project (100 Marks): Week 23 Lab Build: URL Shortener + Sliding-Window Rate Limiter + 120-DSA Portfolio — Complete the timed Backend Machine-Coding Pack (Base62 URL Shortener + Redis Sliding-Window `429` Rate Limiter + Reusable Auth/Order Workflow) alongside your 120-problem Python 3.12 DSA repo...

MODULE 24 OF 24 • STAGE 6 • 25 LIVE + LAB HOURS • 14 TOPICS (112 SUBTOPICS) • 1 CAPSTONE PROJECT (100 MARKS)

Module 24: Final Industry Capstone Project and Placement Readiness

Design, build, test, document, containerize, deploy, and defend the Final Industry Capstone — the Enterprise Job Portal Backend System across all 12 production modules (`Authentication`, `Users`, `Candidates`, `Employers`, `Jobs`, `Search`, `Applications`, `Resume Upload`, `Notifications`, `Admin`, `Reporting`, `Swagger/OpenAPI`) using Python 3.12, FastAPI, DRF fundamentals, PostgreSQL, SQLAlchemy, Redis, Celery, RabbitMQ/Kafka, JWT, Pytest, Docker Compose, and GitHub Actions.

Objectives: ✓ Deliver all 12 independently organized backend modules of the Enterprise Job Portal Backend System with clean Service + Repository separation. • ✓ Enforce Argon2 password security, JWT access + refresh rotation, strict `Candidate`/`Employer`/`Admin` RBAC, and magic-byte PDF resume validation. • ✓ Accelerate job search & analytics with PostgreSQL composite indexes + Redis Cache-Aside while processing email notifications via Celery & RabbitMQ/Kafka. • ✓ Ship complete ER & architecture diagrams, Alembic migrations, OpenAPI/Swagger UI, Postman collection, `>= 80%` Pytest suite, Docker Compose, and GitHub Actions CI. • ✓ Pass the Final Live Architecture Presentation & Line-by-Line Technical Viva demonstrating genuine independent backend engineering competency.

24.1 Capstone Module 1: Authentication Service (Registration, Login, JWT & Refresh Rotation)

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

Subtopics Covered: Email registration • Argon2id password hashing • Enumeration-safe login • `15m` JWT access tokens • Production Edge-Case & Pitfall Guard (Capstone Module 1: Authentication Service (Registration, Login, JWT & Refresh Rotation)) • Hands-On Python Implementation & Verification (Capstone Module 1: Authentication Service (Registration, Login, JWT & Refresh Rotation)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 1

Purpose: Securing every downstream Job Portal route behind verified identity and token rotation.

What Topic Is Next → Next Topic 24.2: Capstone Module 2: User Management & Role-Based Profiles

24.2 Capstone Module 2: User Management & Role-Based Profiles

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

Subtopics Covered: Account profile (`GET/PATCH /users/me`) • Role assignment (`CANDIDATE` • `EMPLOYER` • `ADMIN`) • Production Edge-Case & Pitfall Guard (Capstone Module 2: User Management & Role-Based Profiles) • Hands-On Python Implementation & Verification (Capstone Module 2: User Management & Role-Based Profiles) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 2

Purpose: Enforcing clean role boundaries across Candidate, Employer, and Admin workflows.

What Topic Is Next → Next Topic 24.3: Capstone Module 3: Candidate Profile & Skills Management

24.3 Capstone Module 3: Candidate Profile & Skills Management

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

Subtopics Covered: Headline • Bio • Experience years • Current/expected CTC • Production Edge-Case & Pitfall Guard (Capstone Module 3: Candidate Profile & Skills Management) • Hands-On Python Implementation & Verification (Capstone Module 3: Candidate Profile & Skills Management) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory

Lifecycle for Capstone Module 3

Purpose: Enabling candidates to maintain structured searchable skill profiles (`Python`, `FastAPI`, `PostgreSQL`, `Docker`).

What Topic Is Next → Next Topic 24.4: Capstone Module 4: Employer & Verified Company Profile Management

24.4 Capstone Module 4: Employer & Verified Company Profile Management [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Company onboarding • Slug generation • Website/industry/headquarters metadata • Recruiter ownership • Production Edge-Case & Pitfall Guard (Capstone Module 4: Employer & Verified Company Profile Management) • Hands-On Python Implementation & Verification (Capstone Module 4: Employer & Verified Company Profile Management) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 4

Purpose: Requiring employer verification before published jobs appear on the public candidate search feed.

What Topic Is Next → Next Topic 24.5: Capstone Module 5: Job Lifecycle Management (Draft, Publish, Update, Archive)

24.5 Capstone Module 5: Job Lifecycle Management (Draft, Publish, Update, Archive) [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Job creation (`DRAFT`) • Validation (`max_salary >= min_salary`) • State machine (`DRAFT -> PUBLISHED -> CLOSED / ARCHIVED`) • Employer-scoped management • Production Edge-Case & Pitfall Guard (Capstone Module 5: Job Lifecycle Management (Draft, Publish, Update, Archive)) • Hands-On Python Implementation & Verification (Capstone Module 5: Job Lifecycle Management (Draft, Publish, Update, Archive)) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 5

Purpose: Allowing recruiters to draft, edit, publish, and close job openings safely.

What Topic Is Next → Next Topic 24.6: Capstone Module 6: High-Speed Job Search, Multi-Facet Filtering & Redis Caching

24.6 Capstone Module 6: High-Speed Job Search, Multi-Facet Filtering & Redis Caching [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Keyword search • `city` • `is_remote` • `skills` • Production Edge-Case & Pitfall Guard (Capstone Module 6: High-Speed Job Search, Multi-Facet Filtering & Redis Caching) • Hands-On Python Implementation & Verification (Capstone Module 6: High-Speed Job Search, Multi-Facet Filtering & Redis Caching) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 6

Purpose: Serving `/api/v1/jobs/search` in `< 15ms` under high read concurrency with post-commit cache invalidation on job edits.

What Topic Is Next → Next Topic 24.7: Capstone Module 7: Idempotent Job Application & Status Tracking Workflow

24.7 Capstone Module 7: Idempotent Job Application & Status Tracking Workflow [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: `POST /jobs/{id}/apply` (`UNIQUE (job_id • Candidate_id)` preventing duplicate applications -> `409`) • Cover note • Resume snapshot • Production Edge-Case & Pitfall Guard (Capstone Module 7: Idempotent Job Application & Status Tracking Workflow) • Hands-On Python Implementation & Verification (Capstone Module 7: Idempotent Job Application & Status Tracking Workflow) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 7

Purpose: Ensuring candidates apply once per job and recruiters only view/transition applications belonging to their own company's jobs.

What Topic Is Next → Next Topic 24.8: Capstone Module 8: Secure Resume Upload, Validation & Storage Service

24.8 Capstone Module 8: Secure Resume Upload, Validation & Storage Service [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Chunked `UploadFile` streaming (`<= 5 MB`) • `.pdf`/`.docx` allowlist • `%PDF-` magic-byte header verification • UUID storage key • Production Edge-Case & Pitfall Guard (Capstone Module 8: Secure Resume Upload, Validation & Storage Service) • Hands-On Python Implementation & Verification (Capstone Module 8: Secure Resume Upload, Validation & Storage Service) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 8

Purpose: Allowing ONLY the owning Candidate and the Employer of a job the candidate applied to download that candidate's resume!

What Topic Is Next → Next Topic 24.9: Capstone Module 9: Celery + RabbitMQ/Kafka Background Email Notifications

24.9 Capstone Module 9: Celery + RabbitMQ/Kafka Background Email Notifications [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Post-commit Outbox/on-commit dispatch to Celery + Redis/RabbitMQ/Kafka sending Application Confirmation • Employer New-Applicant Alert • Status-Update emails with retries & DLQ • Production Edge-Case & Pitfall Guard (Capstone Module 9: Celery + RabbitMQ/Kafka Background Email Notifications) • Hands-On Python Implementation & Verification (Capstone Module 9: Celery + RabbitMQ/Kafka Background Email Notifications) • Capstone Module 9: Celery + RabbitMQ/Kafka Background Email Notifications — Technical Interview & Code Walkthrough Readiness • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 9

Purpose: Keeping `POST /apply` sub-30ms while reliably delivering transactional emails with exponential retry and dead-letter protection.

What Topic Is Next → Next Topic 24.10: Capstone Module 10: Admin Moderation & Platform Governance APIs

24.10 Capstone Module 10: Admin Moderation & Platform Governance APIs [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Admin-only endpoints (`/api/v1/admin/*`): verifying companies • Suspending abusive users (`is_active=False`) • Moderating flagged jobs • Viewing audit logs • Production Edge-Case & Pitfall Guard (Capstone Module 10: Admin Moderation & Platform Governance APIs) • Hands-On Python Implementation & Verification (Capstone Module 10: Admin Moderation & Platform Governance APIs) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 10

Purpose: Giving platform administrators safe, audited governance controls.

What Topic Is Next → Next Topic 24.11: Capstone Module 11: Hiring Funnel & Application Reporting Analytics

24.11 Capstone Module 11: Hiring Funnel & Application Reporting Analytics [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: SQL `GROUP BY` / `FILTER` / Window aggregations reporting applications per job • Stage conversion rates (`Applied -> Shortlisted -> Offered`) • Top skills in demand • Monthly trends • Production Edge-Case & Pitfall Guard (Capstone Module 11: Hiring Funnel & Application Reporting Analytics) • Hands-On Python Implementation & Verification (Capstone Module 11: Hiring Funnel & Application Reporting Analytics) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 11

Purpose: Powering Employer Hiring Analytics and Platform Admin KPI dashboards directly from single-pass PostgreSQL aggregations + Redis TTL cache.

What Topic Is Next → Next Topic 24.12: Capstone Module 12: OpenAPI 3 / Swagger UI Specification & Postman Suite

24.12 Capstone Module 12: OpenAPI 3 / Swagger UI Specification & Postman Suite [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Complete tagged OpenAPI 3 documentation (`/docs` & `/redoc`) • Typed request/response/error schemas • ER diagram • Architecture diagram • Production Edge-Case & Pitfall Guard (Capstone Module 12: OpenAPI 3 / Swagger UI Specification & Postman Suite) • Hands-On Python Implementation & Verification (Capstone Module 12: OpenAPI 3 / Swagger UI Specification & Postman Suite) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Module 12

Purpose: Allowing any frontend engineer or technical interviewer to explore and execute every endpoint interactively.

What Topic Is Next → Next Topic 24.13: Capstone Pytest Suite, Multi-Container Docker Compose & GitHub Actions CI

24.13 Capstone Pytest Suite, Multi-Container Docker Compose & GitHub Actions CI [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Unit + Integration `pytest` suite (`>= 80%` coverage across Auth • RBAC • Search • Apply • Production Edge-Case & Pitfall Guard (Capstone Pytest Suite, Multi-Container Docker Compose & GitHub Actions CI) • Hands-On Python Implementation & Verification (Capstone Pytest Suite, Multi-Container Docker Compose & GitHub Actions CI) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Capstone Pytest Suite

Purpose: Proving `docker compose up -d --build` boots the entire platform cleanly and GitHub Actions verifies every PR.

What Topic Is Next → Next Topic 24.14: Final Architecture Presentation, Live API Demo & Line-by-Line Technical Viva

24.14 Final Architecture Presentation, Live API Demo & Line-by-Line Technical Viva [50 mins • Assignment (100M) • Hands-On Lab (100M) • 1 Mock Sets • 1 AI Vivas]

Subtopics Covered: Presenting ER schema • Service/repository architecture • JWT/RBAC security • Redis/Celery/Broker flows • Production Edge-Case & Pitfall Guard (Final Architecture Presentation, Live API Demo & Line-by-Line Technical Viva) • Hands-On Python Implementation & Verification (Final Architecture Presentation, Live API Demo & Line-by-Line Technical Viva) • Idiomatic PBCP Implementation: clean modular architecture, strict contracts & non-blocking patterns • Python Backend Developer – Industry-Ready Career Program Production Runtime Execution & Memory Lifecycle for Final Architecture Presentation

Purpose: Graduating job-ready to explain, debug, extend, and defend your complete Python Backend platform in real software-engineering interviews!

What Topic Is Next → Module 24 Capstone → End-of-Course Live Client Project: Python Backend Developer – Industry-Ready Career Program — Enterprise Client Production Platform

Module 24 Hands-On Capstone Project (100 Marks): Project 6 (Final Capstone): Enterprise Job Portal Backend System — Design, develop, secure, test, document, containerize, automate in CI/CD, deploy, and defend a complete production-grade Enterprise Job Portal Backend System comprising 12 independently orga...

3. LIVE CLIENT PRODUCTION PROJECT & CAREER SCOPE

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

Project: Python Backend Developer – Industry-Ready Career Program — Enterprise Client Production Platform • **Confirmed Role:** Python Backend Developer – Industry-Ready Career Program Engineering Project Intern

Domain & Production Infra: Live Enterprise Python Backend, Django & FastAPI Engineering Client Engagement (Python 3.12 (Core & Advanced), Python OOP & Dataclasses, Decorators, Generators & Iterators, Threading, Multiprocessing & Asyncio, Python DSA (120+ Problems) • Git CI/CD + Automated Testing + Live HTTPS Cloud Production)

- Translated live client functional and non-functional requirements into modular Python 3.12 (Core & Advanced), Python OOP & Dataclasses, Decorators, Generators & Iterators, Threading, Multiprocessing & Asyncio, Python DSA (120+ Problems) 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: Python Backend Developer – Industry-Ready Career Program Developer, Associate Python Backend, Django & FastAPI Engineering 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

Optional Worthwhile Global Certifications (100% Exam-Success Mentorship Available): **OpenJS Node.js Services Developer (JSNSD):** ₹14.0 LPA – ₹30.0 LPA (API & Security Exam) • **MongoDB Certified Associate Developer (Node.js) (C100DEV):** ₹10.0 LPA – ₹24.0 LPA (+25% Data Backend Edge) • **AWS Certified Solutions Architect – Associate (SAA-C03):** ₹11.0 LPA – ₹26.0 LPA (#1 High-ROI Cloud Credential)

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

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