Admissions & Support: support@mockattempt.com
Phone / WhatsApp: +91 98765 43210
Course Code: DSAP • **Updated:** 01 Nov 2026
https://mockattempt.com/courses/dsa-with-python-training

DSAP • DATA STRUCTURES, ALGORITHMS & FAANG CODING PREP | **BEGINNER TO COMPETITIVE & PRODUCT-COMPANY DSA (PYTHON)**

100% LIVE INDUSTRY EXPERT COHORT + INTERNSHIP LETTER

DSA with Python Training

Master product-company DSA & Design and Analysis of Algorithms (DAA) in Python 3.12: Complexity & Master Theorem, bisect Binary Search, Merge/Quick/Heap Sort, CPython Dict Hashing, Red-Black Trees, Memoization vs Tabulation DP, 0/1 Knapsack, LCS, O(N log N) LIS, Greedy Huffman, Backtracking N-Queens, Dijkstra, Bellman-Ford & KMP — with 10×50 MCQ Mock Tests per topic and Voice Gemini AI Interviews at ₹4,999.

Duration: 15–17 Weeks • Live + Hands-On Projects (222 Hrs) • **Modules:** 16 • **Live Topics:** 42 • **Labs & Tasks:** 84 • **Mocks / AI Vivas:** 244 / 244 • **Projects:** 16 Module + 1 Live Client Prod

Fixed Non-Negotiable Tuition Fee: Base Fee **₹4,999** + 18% GST (₹900) = **Total Incl. GST: ₹5,899**

Live Batch Tracks: Weekday Live Online Batch (Mon – Fri) (Monday to Friday (5 Live Interactive Classes / Week) • 1.5 Hours / Day (60 Mins Live Architecture & Coding + 30 Mins Hands-On Lab) • 19–21 Weeks Fast-Track) | Weekend Intensive Live Online Batch (Sat – Sun) (Saturday & Sunday (Designed for Working Professionals & College Students) • 3.0 Hours / Day (2 Hrs Deep-Dive Live Coding + 1 Hr Guided Project Build) • 21–23 Weeks Weekend Track)

Core Technologies & Engineering Stack: Python 3.12 Type-Hinted DSA • Master Theorem & Recurrence Analysis • bisect, heapq & collections.deque • CPython Compact Dict & Red-Black Trees • Dynamic Programming (Knapsack, LCS, LIS) • Greedy, Huffman & Backtracking (N-Queens) • Dijkstra, Bellman-Ford & KMP String Match

1. PROGRAM MASTERY OUTCOMES & ELIGIBILITY

End-to-End Engineering Outcomes:

1. Solve divide-and-conquer recurrences via the Master Theorem and avoid hidden CPython `O(N)` traps (`list.pop(0)`, string `+=` loops, shallow `[[0]*M]*N` aliasing)
2. Implement optimal Binary Search (`bisect_left`), Merge/Quick/Heap Sort, CPython compact-dict hashing, and Red-Black Tree invariants
3. Conquer FAANG Dynamic Programming (`@cache` vs Rolling-Row Tabulation, 1D 0/1 Knapsack, LCS, `O(N log N)` Patience LIS), Greedy Huffman Trees, and N-Queens Backtracking
4. Clear 200 Timed Mock Exams (10,000 Python DSA MCQs) and 200 Spoken Voice + Gemini AI Algorithmic Viva Rounds
5. Deep understanding of CPython complexity (`list` vs `collections.deque`, `dict` hashing, `heapq`, `bisect`, `Counter`).
6. Pattern-driven mastery: Sliding Window, Two Pointers, Fast/Slow Pointers, Prefix Sums, Binary Search on Answer, and Monotonic Stacks.
7. Trees, Tries, Heaps, Backtracking pruning, Graph traversals/shortest paths, Union-Find, and 1D/2D Dynamic Programming.
8. Structured 4-step live interview communication: clarify constraints → brute-force baseline → optimal Big-O → dry-run edge cases.

Eligibility & Prerequisites: Basic familiarity with Python 3 syntax (lists, dicts, loops, functions) — recursion, complexity math, and stdlib DSA modules are taught step-by-step • Any Windows, macOS, or Linux laptop running Python 3.11/3.12 and VS Code or PyCharm Community • No prior DSA with Python industry experience required — Module 1 starts from environment setup, core mental models, and plain-English fundamentals. • Basic computer & internet familiarity on a Windows, macOS, or Linux laptop (8 GB+ RAM recommended for smooth live coding labs). • Ideal for B.Tech / BCA / MCA / B.Sc IT students, non-CS beginners, career switchers, and working developers upgrading to 2026 production standards. • Commitment to attend live Weekday or Weekend classes, complete daily hands-on lab checklists, and practice topic-wise Mock Tests & AI Interviews.

2. DETAILED MODULE & TOPIC-WISE SYLLABUS (16 MODULES • 42 LIVE TOPICS)

MODULE 1 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 1: CPython Complexity Internals & Divide-and-Conquer Master Theorem

Analyze exact CPython container time/space complexities (`list`, `deque`, `set`, `dict`, slicing costs) and solve `T(n) = aT(n/b) + f(n)` divide-and-conquer recurrences instantaneously via the Master Theorem.

Objectives: ✓ Spot hidden CPython `O(N)` bottlenecks (`x in list`, `list.pop(0)`, `arr[:k]` slice copies) and replace them with `O(1)` structures • ✓ Avoid Python's classic `[[0] * cols] * rows` shallow row-aliasing bug when initializing 2D DP/Grid matrices • ✓ Classify `T(n) = aT(n/b) + f(n)` into Case 1 (leaf-heavy),

Case 2 (balanced levels), and Case 3 (root-heavy + regularity check)

1.1 Algorithm Time & Space Complexity in Python — CPython Stdlib Costs & Hidden `O(N)` Traps [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Algorithm Time & Space Complexity in Python — CPython Stdlib Costs & Hidden `O(N)` Traps in real DSA with Python systems — specifically: Understan...

1.2 Master Theorem (`T(n) = aT(n/b) + f(n)`) — Critical Exponent `n^(log_b a)` & Cases 1, 2, 3 [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Master Theorem (`T(n) = aT(n/b) + f(n)`) — Critical Exponent `n^(log_b a)` & Cases 1, 2, 3 in real DSA with Python systems — specifically: Solve r...

Module 1 Hands-On Capstone Project (100 Marks): Automated Master Theorem Recurrence Classifier & CPython Complexity Auditor — Write a Python 3.12 solver classifying `T(n) = a * T(n / b) + n^k * (log n)^p` across Cases 1, 2, and 3 by comparing `c_crit = log_b(a)` against `k`, paired with a `deque` vs `list.pop(0)` o...

MODULE 2 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 2: Python `bisect` Binary Search & Guaranteed `O(N log N)` Merge Sort

Leverage C-accelerated `bisect_left` / `bisect_right` boundary searches and monotonic parametric Binary Search, and implement index-bounded `O(N log N)` Merge Sort without `O(N log N)` slice-copy overhead.

Objectives: ✓ Use `bisect_left` (first index `>= x`) and `bisect_right` (first index `> x`) to count range occurrences in `O(log N)` • ✓ Implement index-passed Merge Sort `(lo, hi)` instead of recursive `arr[:mid]` list slicing cleanly • ✓ Understand how Python’s built-in `list.sort()` / `sorted()` **Timsort** achieves `O(N)` best-case speed on partially ordered data

2.1 Binary Search in Python — `bisect_left` vs `bisect_right` & Parametric Search on Answer [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Binary Search in Python — `bisect_left` vs `bisect_right` & Parametric Search on Answer in real DSA with Python systems — specifically: Master C-i...

2.2 Merge Sort in Python — Index-Bounded Divide & Conquer, Stability & CPython Timsort (`O(N)` Best Case) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Merge Sort in Python — Index-Bounded Divide & Conquer, Stability & CPython Timsort (`O(N)` Best Case) in real DSA with Python systems — specifical...

Module 2 Hands-On Capstone Project (100 Marks): `bisect` Range-Frequency Query Engine & Index-Bounded `O(N log N)` Merge Sort — Build an `O(log N)` sorted range-count query responder (`count_in_range(arr, low_val, high_val) = bisect_right(arr, high_val) - bisect_left(arr, low_val)`) paired with a stable Merge Sort co...

MODULE 3 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 3: Randomized 3-Way `Quick Sort` / Quickselect & In-Place `Binary Heap Sort` (`heapq`)

Implement duplicate-proof Randomized 3-Way Quick Sort + `O(N)` Quickselect, and master `O(1)`-space in-place Binary Heap Sort alongside Python’s `heapq` (`heapify`, `heappushpop`, `nlargest`).

Objectives: ✓ Eliminate both sorted-input and all-duplicate `O(N^2)` Quick Sort traps using `random.randint` + 3-Way Dutch National Flag partitioning • ✓ Implement `O(1)` auxiliary-space Heap Sort by building a Max-Heap bottom-up in `O(N)` and extracting to the array tail • ✓ Use Python’s Min-Heap `heapq` cleanly (including `-val` negation or wrapper tuples for Max-Heap priority queues)

3.1 Quick Sort & Quickselect — Randomized Pivot, 3-Way Partitioning & Python Recursion Depth Guard [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Quick Sort & Quickselect — Randomized Pivot, 3-Way Partitioning & Python Recursion Depth Guard in real DSA with Python systems — specifically: Und...

3.2 Binary Heap Sort & Python `heapq` — `O(N)` Sift-Down `heapify`, `O(1)` Space Sort & Min-Heap Idioms [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Binary Heap Sort & Python `heapq` — `O(N)` Sift-Down `heapify`, `O(1)` Space Sort & Min-Heap Idioms in real DSA with Python systems — specifically...

Module 3 Hands-On Capstone Project (100 Marks): Randomized 3-Way Quickselect (`O(N)` `K`-th Element) & `O(1)`-Space In-Place Heap Sort — Build an `O(N)` expected-time Quickselect solver handling 100,000 duplicate values without `RecursionError` (iteratively narrowing `(lo, hi)` around the 3-way `(lt, gt)` pivot band), paired...

MODULE 4 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 4: CPython Open-Addressing `dict` Hashing & Self-Balancing Red-Black Trees

Explore CPython 3.6+ compact insertion-ordered `dict` internals (open addressing, perturbed probing, tombstones, SipHash DoS randomization) and master the 5 invariants of **Red-Black Trees** (`<= 2 log2(N+1)` height).

Objectives: ✓ Explain how CPython's compact `dict` stores a sparse index table + dense insertion-ordered entry array using 2/3 load factor • ✓ Understand why `list` / `set` / `dict` are unhashable (`TypeError: unhashable type`) while `tuple`, `frozenset`, and `@dataclass(frozen=True)` work as keys • ✓ Verify all 5 Red-Black Tree invariants (root black, red children black, equal black-height on all root-to-NIL paths) and compare RBT vs AVL

4.1 Hashing & CPython `dict` Internals — Compact Open Addressing, Tombstones & `frozenset`/Tuple Keys [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Hashing & CPython `dict` Internals — Compact Open Addressing, Tombstones & `frozenset`/Tuple Keys in real DSA with Python systems — specifically:...

4.2 Red-Black Tree — 5 Color Invariants, Black-Height, Recoloring vs Rotations & `O(1)` Fixup Bound [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Red-Black Tree — 5 Color Invariants, Black-Height, Recoloring vs Rotations & `O(1)` Fixup Bound in real DSA with Python systems — specifically: Un...

Module 4 Hands-On Capstone Project (100 Marks): Open-Addressing Linear/Perturbed Hash Table & Red-Black Tree Invariant Verifier — Implement an open-addressed hash map handling collisions and `TOMBSTONE` dummy deletions without breaking probe chains, paired with a Red-Black Tree property verifier computing exact black-h...

MODULE 5 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 5: Dynamic Programming Foundations — Optimal Substructure & `@cache` Memoization vs Tabulation

Transition smoothly from exponential `O(2^N)` brute-force recursion to polynomial Dynamic Programming: identify **Overlapping Subproblems + Optimal Substructure**, and master Top-Down `@functools.cache` vs Space-Optimized Bottom-Up Tabulation.

Objectives: ✓ Formulate any DP problem systematically via the 4-step framework: State Definition -> Base Cases -> Transition Recurrence -> Evaluation Order • ✓ Compare Top-Down Memoization (`@functools.cache`) against Bottom-Up Tabulation (zero recursion overhead + rolling-variable `O(1)` space compression) • ✓ Know when Top-Down wins (sparse reachable state space) vs when Bottom-Up wins (`N = 10^5` exceeding Python's recursion limit or needing rolling-row memory compression)

5.1 Dynamic Programming — Overlapping Subproblems, Optimal Substructure & State Compression [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Dynamic Programming — Overlapping Subproblems, Optimal Substructure & State Compression in real DSA with Python systems — specifically: Understand...

5.2 Tabulation vs Memoization — `@functools.cache` Top-Down vs Iterative Rolling-Row Bottom-Up [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Tabulation vs Memoization — `@functools.cache` Top-Down vs Iterative Rolling-Row Bottom-Up in real DSA with Python systems — specifically: Compare...

Module 5 Hands-On Capstone Project (100 Marks): Coin Change Minimum Coins & Rolling-State House Robber DP Engine — Solve Unbounded Coin Change (`min coins to make amount`, returning `-1` if unreachable) via Bottom-Up Tabulation alongside a space-compressed `O(1)`-memory state-machine DP (`House Robber` /...

MODULE 6 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 6: Classic 2D & Space-Compressed DP — `0/1 Knapsack` & `Longest Common Subsequence (LCS)`

Master pseudo-polynomial **0/1 Knapsack** (and why iterating `w` BACKWARDS from `W` down to `wt` compresses space to `O(W)`!) and **Longest Common Subsequence (LCS)** string/diff alignment + backpointer reconstruction.

Objectives: ✓ Distinguish **0/1 Knapsack** (reverse inner loop `range(W, wt - 1, -1)`) from **Unbounded Knapsack** (forward inner loop `range(wt, W + 1)`) • ✓ Understand why `O(N * W)` is **pseudo-polynomial** (exponential in the bit-length `log2(W)` of input capacity `W`) • ✓ Solve LCS (`dp[i][j]`) in `O(N * M)` time, compress value computation to `O(min(N, M))` space, and reconstruct the actual subsequence string

6.1 0/1 Knapsack — Reverse-Capacity `O(W)` 1D Compression, Subset Sum & Pseudo-Polynomial Time [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply 0/1 Knapsack — Reverse-Capacity `O(W)` 1D Compression, Subset Sum & Pseudo-Polynomial Time in real DSA with Python systems — specifically: Maximiz...

6.2 Longest Common Subsequence (LCS) — 2-String Alignment DP, Backtracking & `O(min(N, M))` Space [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Longest Common Subsequence (LCS) — 2-String Alignment DP, Backtracking & `O(min(N, M))` Space in real DSA with Python systems — specifically: Alig...

Module 6 Hands-On Capstone Project (100 Marks): 1D Space-Optimized 0/1 Knapsack & Git-Style LCS Diff Reconstructor — Implement a 1D `O(W)`-space 0/1 Knapsack solver alongside Subset-Sum Partitioning, plus an LCS DP engine that reconstructs the exact longest common subsequence string (basis of `git diff`) v...

MODULE 7 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 7: `O(N log N)` Longest Increasing Subsequence (LIS) & Greedy Exchange Proofs

Accelerate Longest Increasing Subsequence from `O(N^2)` DP to `O(N log N)` **Patience Sorting + `bisect_left`**, and master **Greedy Algorithms** (Greedy Choice Property, Exchange Argument, Interval Scheduling, Jump Game).

Objectives: ✓ Upgrade `O(N^2)` LIS DP (`N = 2000`) to `O(N log N)` minimal-tail array + `bisect_left` (`N = 10^5`, including 2D Russian Doll Envelopes) • ✓ Know why the `tails` array in Patience LIS has the exact right **length**, even though `tails` itself is NOT necessarily a valid subsequence! • ✓ Prove Greedy optimality via the **Exchange Argument** (sorting intervals by **earliest finish time**, NOT earliest start time or shortest duration!)

7.1 Longest Increasing Subsequence (LIS) — `O(N^2)` DP vs `O(N log N)` Patience `tails` + `bisect_left` [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Longest Increasing Subsequence (LIS) — `O(N^2)` DP vs `O(N log N)` Patience `tails` + `bisect_left` in real DSA with Python systems — specifically...

7.2 Greedy Algorithms — Greedy Choice Property, Exchange Argument, Interval Scheduling & Fractional Knapsack [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Greedy Algorithms — Greedy Choice Property, Exchange Argument, Interval Scheduling & Fractional Knapsack in real DSA with Python systems — specifi...

Module 7 Hands-On Capstone Project (100 Marks): `O(N log N)` 2D Russian-Doll Envelope LIS & Earliest-Finish Interval Scheduler — Solve 2D Envelope Nesting (`w1 < w2` and `h1 < h2`) in `O(N log N)` by sorting `(width ASC, -height DESC)` + Patience LIS on heights, paired with an optimal Greedy Non-Overlapping Activity S...

MODULE 8 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 8: Prefix-Free Greedy `Huffman Coding` & State-Space `Backtracking` Pruning

Build optimal lossless prefix-free binary codes in `O(K log K)` using a Min-Heap **Huffman Tree**, and master exhaustive search with aggressive constraint pruning via the **Choose -> Explore -> Unchoose** Backtracking template.

Objectives: ✓ Construct variable-length prefix-free Huffman bitstrings via `heapq` merging of the two lowest-frequency symbols • ✓ Explain why Huffman's **Prefix-Free Property** (symbols live exclusively at leaf nodes!) guarantees unambiguous bit-stream decoding without delimiters • ✓ Apply the canonical **Choose -> Explore -> Unchoose (pop)** Backtracking pattern across Subsets, Permutations, Combination Sum, and Word Search

8.1 Huffman Coding — Optimal Prefix-Free Binary Tree, `heapq` Merging & Compression Bit Ratio [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Huffman Coding — Optimal Prefix-Free Binary Tree, `heapq` Merging & Compression Bit Ratio in real DSA with Python systems — specifically: Assign s...

8.2 Backtracking — State-Space Tree, `Choose -> Explore -> Unchoose`, Pruning & Duplicate Skipping [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Backtracking — State-Space Tree, `Choose -> Explore -> Unchoose`, Pruning & Duplicate Skipping in real DSA with Python systems — specifically: Exp...

Module 8 Hands-On Capstone Project (100 Marks): Lossless Huffman Bit-Encoder/Decoder & Pruned Combination-Sum Backtracker — Implement a complete Huffman encoder + tree-walking bit decoder handling both multi-symbol strings and single-unique-character edge cases (`"AAAAA"`), paired with a sorted early-break Combin...

MODULE 9 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 9: Constraint-Satisfaction `N-Queens` (`O(1)` Diagonal Sets) & Weighted `Dijkstra's Algorithm`

Slash N-Queens conflict checks from `O(N)` board scans to `O(1)` diagonal invariants (`r - c` and `r + c`) and compute single-source weighted shortest paths in `O((V + E) log V)` with lazy-heap **Dijkstra's Algorithm**.

Objectives: ✓ Place `N` non-attacking Queens row-by-row using `cols`, `diag1 (r - c)`, and `diag2 (r + c)` `O(1)` lookup sets (or integer bitmasks) • ✓ Implement lazy Min-Heap Dijkstra in Python with the critical `if d > dist[u]: continue` stale-entry skip • ✓ Explain precisely why Dijkstra FAILS on graphs containing negative edge weights (greedy finalization invariant breaks!)

9.1 N-Queens Problem — Row-by-Row Placement & `O(1)` Diagonal Invariants (`r - c` and `r + c`) [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply N-Queens Problem — Row-by-Row Placement & `O(1)` Diagonal Invariants (`r - c` and `r + c`) in real DSA with Python systems — specifically: Place o...

9.2 Dijkstra's Algorithm — Non-Negative Edge Relaxation, Lazy `heapq` & Stale-Entry Guard [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Dijkstra's Algorithm — Non-Negative Edge Relaxation, Lazy `heapq` & Stale-Entry Guard in real DSA with Python systems — specifically: Compute sing...

Module 9 Hands-On Capstone Project (100 Marks): Bitmask-Accelerated `N-Queens` Solver & Path-Reconstructing `Dijkstra` Router — Implement an `O(1)`-conflict N-Queens solver counting distinct solutions for `N = 1..10`, paired with a `heapq` Dijkstra router returning both minimum latency and the exact hop-by-hop shorte...

MODULE 10 OF 16 • 13 LIVE + LAB HOURS • 2 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Module 10: Negative-Weight `Bellman-Ford` Cycle Detection & Linear `O(N + M)` `KMP` String Matching

Handle negative edge weights, currency-arbitrage negative cycles, and `<= K`-stop flight constraints with **Bellman-Ford (`O(V * E)`)**, and search substrings in strict worst-case `O(N + M)` linear time using **Knuth-Morris-Pratt (KMP)** `LPS` failure tables.

Objectives: ✓ Relax all `E` edges `V - 1` times in Bellman-Ford, early-terminate when no distance improves, and run a `V`-th check to detect reachable negative-weight cycles • ✓ Snapshot `prev_dist = dist.copy()` per round when solving `Cheapest Flights Within K Stops` so multi-hop cascades don't sneak into 1 round • ✓ Build the KMP Longest Proper Prefix which is also Suffix (`LPS` / `pi`) table in `O(M)` and find all pattern matches in `O(N + M)` without ever rewinding text pointer `i`

10.1 Bellman-Ford Algorithm — `V - 1` Edge Relaxations, Negative Cycle Detection & `K`-Stop Flights [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Bellman-Ford Algorithm — `V - 1` Edge Relaxations, Negative Cycle Detection & `K`-Stop Flights in real DSA with Python systems — specifically: Why...

10.2 Knuth-Morris-Pratt (KMP) — `LPS` Prefix-Suffix Automaton & Worst-Case `O(N + M)` Pattern Search [50 mins • Assignment (100M) • Hands-On Lab (50M) • 10 Mock Sets • 10 AI Vivas]

Purpose: To understand and apply Knuth-Morris-Pratt (KMP) — `LPS` Prefix-Suffix Automaton & Worst-Case `O(N + M)` Pattern Search in real DSA with Python systems — specifically: El...

Module 10 Hands-On Capstone Project (100 Marks): Negative-Cycle Arbitrage Detector (`Bellman-Ford`) & `O(N + M)` Multi-Match `KMP` Scanner — Build a Bellman-Ford shortest-path & negative-cycle detector supporting bounded `K`-edge relaxations via snapshot arrays, paired with a full Knuth-Morris-Pratt (`LPS` builder + streaming mat...

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

Advanced Graph Algorithms in Python: deque, heapq, Topo Sort & DSU

Production-grade DSA with Python Training engineering module covering Advanced Graph Algorithms in Python: deque, heapq, Topo Sort & DSU with hands-on labs and capstone build.

Objectives: ✓ collections.deque BFS, Iterative DFS & Recursion-Limit Safety — Purpose: Avoid Python RecursionError crashes on 10^5-node graphs using explicit stacks/deques. • ✓ Dijkstra with heapq, 0-1 BFS & Shortest Path with State Tuples — Purpose: Solve constrained shortest-path OA problems cleanly using tuple priority queues. • ✓ Topological Sort (Kahn's Algorithm) & Disjoint Set Union (Union-Find) — Purpose: Detect dependency cycles and merge dynamic connected components in O(alpha(N)). • ✓ Multi-Source Grid BFS & Implicit Word/State-Space Graphs — Purpose: Crack shortest transformation and grid wave-propagation problems rapidly.

11.1 collections.deque BFS, Iterative DFS & Recursion-Limit Safety [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply collections.deque BFS, Iterative DFS & Recursion-Limit Safety in real DSA with Python systems — specifically: Avoid Python RecursionError crashes...

11.2 Dijkstra with heapq, 0-1 BFS & Shortest Path with State Tuples [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Dijkstra with heapq, 0-1 BFS & Shortest Path with State Tuples in real DSA with Python systems — specifically: Solve constrained shortest-path OA...

11.3 Topological Sort (Kahn's Algorithm) & Disjoint Set Union (Union-Find) [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Topological Sort (Kahn's Algorithm) & Disjoint Set Union (Union-Find) in real DSA with Python systems — specifically: Detect dependency cycles and...

11.4 Multi-Source Grid BFS & Implicit Word/State-Space Graphs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Multi-Source Grid BFS & Implicit Word/State-Space Graphs in real DSA with Python systems — specifically: Crack shortest transformation and grid wa...

Module 11 Hands-On Capstone Project (100 Marks): High-Speed Network Routing & Dependency Graph Solver in Python — Design, implement, test, and document 'High-Speed Network Routing & Dependency Graph Solver in Python' applying every production concept from Module 11 (Advanced Graph Algorithms in Python:...

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

Dynamic Programming Mastery in Python: @cache, Knapsack, LIS & Trees

Production-grade DSA with Python Training engineering module covering Dynamic Programming Mastery in Python: @cache, Knapsack, LIS & Trees with hands-on labs and capstone build.

Objectives: ✓ functools.cache Top-Down -> Iterative Bottom-Up -> 1D Rolling Arrays — Purpose: Prototype recurrence relations in 3 minutes and optimize memory to pass strict OA limits. • ✓ 0/1 & Unbounded Knapsack, Partition & Target-Sum Patterns — Purpose: Identify state variables and base cases immediately under timed test pressure. • ✓ LCS, Edit Distance, bisect_left O(N log N) LIS & Interval DP — Purpose: Combine Python bisect binary search with DP for optimal subsequence runtimes. • ✓ Tree DP, Bitmask DP (N <= 20) & Stock Cooldown State Machines — Purpose: Conquer hard-tier product-company OA problems with clean bitwise transitions.

12.1 functools.cache Top-Down -> Iterative Bottom-Up -> 1D Rolling Arrays [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply functools.cache Top-Down -> Iterative Bottom-Up -> 1D Rolling Arrays in real DSA with Python systems — specifically: Prototype recurrence relation...

12.2 0/1 & Unbounded Knapsack, Partition & Target-Sum Patterns [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply 0/1 & Unbounded Knapsack, Partition & Target-Sum Patterns in real DSA with Python systems — specifically: Identify state variables and base cases...

12.3 LCS, Edit Distance, bisect_left O(N log N) LIS & Interval DP [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply LCS, Edit Distance, bisect_left O(N log N) LIS & Interval DP in real DSA with Python systems — specifically: Combine Python bisect binary search w...

12.4 Tree DP, Bitmask DP (N <= 20) & Stock Cooldown State Machines [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Tree DP, Bitmask DP (N <= 20) & Stock Cooldown State Machines in real DSA with Python systems — specifically: Conquer hard-tier product-company OA...

Module 12 Hands-On Capstone Project (100 Marks): Sequence, Knapsack & State-Machine DP Toolkit in Python — Design, implement, test, and document 'Sequence, Knapsack & State-Machine DP Toolkit in Python' applying every production concept from Module 12 (Dynamic Programming Mastery in Python: @cach...

MODULE 13 OF 16 • 16 LIVE + LAB HOURS • 4 TOPICS • 1 CAPSTONE PROJECT (100 MARKS)

Tries, Segment/Fenwick Trees, String Hashing & Sweep-Line Intervals

Production-grade DSA with Python Training engineering module covering Tries, Segment/Fenwick Trees, String Hashing & Sweep-Line Intervals with hands-on labs and capstone build.

Objectives: ✓ Nested-Dict / Node Tries & Word-Search Pruning — Purpose: Implement fast prefix autocomplete and prune exponential board backtracking searches. • ✓ Fenwick Tree (Binary Indexed Tree) & Iterative Segment Tree — Purpose: Support O(log N) dynamic prefix sums and range queries without TLE in Python. • ✓ Polynomial Rolling Hash, KMP & Z-Array Linear String Matching — Purpose: Find repeated substrings and palindromes in guaranteed O(N) linear time. • ✓ Interval Sweep-Line, Prefix Difference Arrays & bisect Event Queries — Purpose: Solve calendar booking, CPU scheduling, and overlapping skylines cleanly.

13.1 Nested-Dict / Node Tries & Word-Search Pruning [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Nested-Dict / Node Tries & Word-Search Pruning in real DSA with Python systems — specifically: Implement fast prefix autocomplete and prune expone...

13.2 Fenwick Tree (Binary Indexed Tree) & Iterative Segment Tree [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Fenwick Tree (Binary Indexed Tree) & Iterative Segment Tree in real DSA with Python systems — specifically: Support O(log N) dynamic prefix sums a...

13.3 Polynomial Rolling Hash, KMP & Z-Array Linear String Matching [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Polynomial Rolling Hash, KMP & Z-Array Linear String Matching in real DSA with Python systems — specifically: Find repeated substrings and palindr...

13.4 Interval Sweep-Line, Prefix Difference Arrays & bisect Event Queries [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Interval Sweep-Line, Prefix Difference Arrays & bisect Event Queries in real DSA with Python systems — specifically: Solve calendar booking, CPU s...

Module 13 Hands-On Capstone Project (100 Marks): Prefix Search & Streaming Range-Query Engine in Python — Design, implement, test, and document 'Prefix Search & Streaming Range-Query Engine in Python' applying every production concept from Module 13 (Tries, Segment/Fenwick Trees, String Hashing...

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

Monotonic Deques, Multi-Heap Streams & Python LLD Machine Coding

Production-grade DSA with Python Training engineering module covering Monotonic Deques, Multi-Heap Streams & Python LLD Machine Coding with hands-on labs and capstone build.

Objectives: ✓ Monotonic Stack & collections.deque Sliding Window Maximum — Purpose: Achieve strict O(N) amortized window scans where naive slicing O(N*K) times out. • ✓ Two-Heap Streaming Median, K-Way Merge & Counter/heapq Top-K — Purpose: Maintain real-time percentiles and merge sorted log streams effortlessly. • ✓ OrderedDict / Custom Doubly-Linked O(1) LRU/LFU & Time-Based KV Store — Purpose: Build clean, extensible classes for 60-minute machine-coding interviews. • ✓ Python Constant-Factor Speedups & Interview Verbalization Framework — Purpose: Avoid list.pop(0) O(N) traps, prove Big-O clearly, and test corner cases first.

14.1 Monotonic Stack & collections.deque Sliding Window Maximum [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Monotonic Stack & collections.deque Sliding Window Maximum in real DSA with Python systems — specifically: Achieve strict O(N) amortized window sc...

14.2 Two-Heap Streaming Median, K-Way Merge & Counter/heapq Top-K [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Two-Heap Streaming Median, K-Way Merge & Counter/heapq Top-K in real DSA with Python systems — specifically: Maintain real-time percentiles and me...

14.3 OrderedDict / Custom Doubly-Linked O(1) LRU/LFU & Time-Based KV Store [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply OrderedDict / Custom Doubly-Linked O(1) LRU/LFU & Time-Based KV Store in real DSA with Python systems — specifically: Build clean, extensible clas...

14.4 Python Constant-Factor Speedups & Interview Verbalization Framework [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Python Constant-Factor Speedups & Interview Verbalization Framework in real DSA with Python systems — specifically: Avoid list.pop(0) O(N) traps,...

Module 14 Hands-On Capstone Project (100 Marks): In-Memory LRU Cache, Rate Limiter & Job Scheduler in Python — Design, implement, test, and document 'In-Memory LRU Cache, Rate Limiter & Job Scheduler in Python' applying every production concept from Module 14 (Monotonic Deques, Multi-Heap Streams & P...

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

Advanced Pythonic Graph, Trie, Union-Find & Range-Query Algorithms

Self-explanatory list-and-bullet production mastery module covering Advanced Pythonic Graph, Trie, Union-Find & Range-Query Algorithms in DSA with Python Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken Gemini AI Viva.

Objectives: ✓ Trie Prefix Trees, Rolling Hash / KMP & Counter/Deque Sliding-Window Idioms — Purpose: Write concise, bug-free Python string and stream algorithms using collections.deque and defaultdict. • ✓ Union-Find (Rank + Path Compression), Shortest Paths (0-1 BFS, Dijkstra) & DAG DP — Purpose: Solve multi-source grid, network latency, and scheduling dependency problems optimally. • ✓ Binary Search on Answer, Monotonic Stacks & Bitmask Dynamic Programming — Purpose: Crack hard optimization and subset-state problems using functools.cache and clean state transitions. • ✓ Build, verify, and document 'High-Speed Python Autocomplete Trie, Union-Find & Fenwick Tree Library' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + Google Gemini AI Technical Interviews.

15.1 Trie Prefix Trees, Rolling Hash / KMP & Counter/Deque Sliding-Window Idioms [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Trie Prefix Trees, Rolling Hash / KMP & Counter/Deque Sliding-Window Idioms in real DSA with Python systems — specifically: Write concise, bug-fre...

15.2 Union-Find (Rank + Path Compression), Shortest Paths (0-1 BFS, Dijkstra) & DAG DP [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Union-Find (Rank + Path Compression), Shortest Paths (0-1 BFS, Dijkstra) & DAG DP in real DSA with Python systems — specifically: Solve multi-sour...

15.3 Binary Search on Answer, Monotonic Stacks & Bitmask Dynamic Programming [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Binary Search on Answer, Monotonic Stacks & Bitmask Dynamic Programming in real DSA with Python systems — specifically: Crack hard optimization an...

Module 15 Hands-On Capstone Project (100 Marks): High-Speed Python Autocomplete Trie, Union-Find & Fenwick Tree Library — Architect, implement, test, and document 'High-Speed Python Autocomplete Trie, Union-Find & Fenwick Tree Library' applying every bullet-wise engineering concept from Module 15 (Advanced Pyth...

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

Python Runtime Pitfalls, Stress Testing & Live Coding Interview Execution

Self-explanatory list-and-bullet production mastery module covering Python Runtime Pitfalls, Stress Testing & Live Coding Interview Execution in DSA with Python Training, paired with hands-on labs, capstone repository, Topic Mock Tests, and Spoken Gemini AI Viva.

Objectives: ✓ Python Performance Rules: Avoiding O(N) list.pop(0), String Concatenation & Recursion Limits — Purpose: Prevent Time Limit Exceeded (TLE) verdicts caused by hidden quadratic operations in Python. • ✓ Property/Stress Testing: Generating Random Counterexamples to Catch Edge-Case Bugs — Purpose: Prove your optimal algorithm matches brute-force output across 1,000+ tricky boundary inputs. • ✓ FAANG / Product-Company Live Coding Framework & Spoken Complexity Walkthroughs — Purpose: Clarify constraints, dry-run examples aloud, and write clean typed Python under 35-minute timers. • ✓ Build, verify, and document 'Randomized Brute-Force vs Optimal Stress-Test Harness in Python 3.12' with clean Git commits and edge-case protection. • ✓ Clear 100% of Topic Timed Mock Tests and Spoken Voice + Google Gemini AI Technical Interviews.

16.1 Python Performance Rules: Avoiding O(N) list.pop(0), String Concatenation & Recursion Limits [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Python Performance Rules: Avoiding O(N) list.pop(0), String Concatenation & Recursion Limits in real DSA with Python systems — specifically: Preve...

16.2 Property/Stress Testing: Generating Random Counterexamples to Catch Edge-Case Bugs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply Property/Stress Testing: Generating Random Counterexamples to Catch Edge-Case Bugs in real DSA with Python systems — specifically: Prove your opti...

16.3 FAANG / Product-Company Live Coding Framework & Spoken Complexity Walkthroughs [65 mins • Assignment (100M) • Hands-On Lab (50M) • 2 Mock Sets • 2 AI Vivas]

Purpose: To understand and apply FAANG / Product-Company Live Coding Framework & Spoken Complexity Walkthroughs in real DSA with Python systems — specifically: Clarify constraints...

Module 16 Hands-On Capstone Project (100 Marks): Randomized Brute-Force vs Optimal Stress-Test Harness in Python 3.12 — Architect, implement, test, and document 'Randomized Brute-Force vs Optimal Stress-Test Harness in Python 3.12' applying every bullet-wise engineering concept from Module 16 (Python Runtime...

3. LIVE CLIENT PRODUCTION PROJECT, GLOBAL CERTIFICATIONS & CAREER SCOPE

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

Project: PyGraph Fraud Ring Detector, Sliding-Window Rate Limiter & Priority Scheduler • **Confirmed Role:** Python Data Structures, Algorithms & Problem-Solving Intern

Domain & Production Infra: FinTech Risk & Real-Time Transaction Screening Team (Python 3.12 + Custom Graph/Union-Find/Heap/Deque Engines + Pytest Complexity Suite)

- Built Disjoint-Set (Union-Find) and BFS/DFS cycle-detection algorithms to flag connected fraud rings in real time.
- Implemented O(1) deque sliding-window rate limiters, monotonic queues, and heapq priority job schedulers.
- Optimized recursive backtracking and dynamic programming solvers using memoization and bottom-up tabulation.
- Validated asymptotic runtime and worst-case adversarial inputs with automated pytest benchmark suites.

Aligned Global Vendor Certifications: **CCDSA-ADV** (Certified Data Structures & Algorithms Proficiency Exam) • **PCAP-31-03** (PCAP™ – Certified Associate in Python Programming)

Target Job Roles: Software Development Engineer — SDE-1 (Python), Backend & Systems Product Engineer, AI / ML Algorithms Engineer, Platform Software Engineer

Realistic Annual CTC Scope: **Product Fresher / SDE-1 (0–1 Yr):** ₹8.0 LPA – ₹18.0 LPA **Product SDE-2 / AI Engineer (1–3 Yrs):** ₹16.0 LPA – ₹32.0 LPA **Senior SDE / Principal Python Architect (3–6+ Yrs):** ₹30.0 LPA – ₹58.0+ LPA

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

Try 2 Free Topic-Wise 50-MCQ Mock Tests & Spoken Gemini AI Interviews daily on every topic above before enrolling.