**1. Strings – Topics List**

**Section 1: Foundation**

1. Introduction to Strings – Definition & Use Cases
2. Creating and Accessing Strings *(Indexing, Negative Indexing)*
3. String Slicing and Substrings
4. String Immutability Explained
5. Basic String Operations: Concatenation, Repetition

**Section 2: Manipulation & Built-in Functions**

1. Case Manipulation (upper(), lower(), title(), capitalize())
2. Stripping & Trimming (strip(), lstrip(), rstrip())
3. Searching & Counting (find(), index(), count())
4. Replacing and Splitting (replace(), split(), join())
5. String Formatting (format(), f-strings, % formatting)

**Section 3: Advanced String Handling**

1. Checking Conditions (isalnum(), isalpha(), isdigit(), etc.)
2. Regular Expressions with re Module
3. Encoding and Decoding Strings (UTF-8, ASCII)
4. String Template Class
5. Performance Considerations in Large Text Processing

**Section 4: Best Practices**

1. Writing Clean, Readable String Code
2. Avoiding Common String Anti-Patterns
3. Hands-On Challenges and Mini Project *(e.g., text analyzer)*

**2. Lists – Topics List**

**Section 1: Foundation**

1. Introduction to Lists – Basics and Use Cases
2. Creating, Indexing, and Slicing Lists
3. Adding Elements (append(), insert(), extend())
4. Removing Elements (remove(), pop(), clear())
5. List Mutability Explained

**Section 2: Built-in Functions & Operations**

1. Searching in Lists (in, index(), count())
2. Sorting and Reversing (sort(), sorted(), reverse())
3. Copying Lists (copy(), deepcopy())
4. List Comprehensions *(Pythonic Loops)*
5. Iterating Over Lists with enumerate()

**Section 3: Advanced Patterns**

1. Nested Lists and Matrix Representations
2. Filtering Data with List Comprehensions
3. Using map(), filter(), and reduce() with Lists
4. List Performance Optimization
5. Avoiding Common Pitfalls *(e.g., modifying while iterating)*

**3. Tuples – Topics List**

**Section 1: Foundation**

1. Introduction to Tuples – Immutable Data Structures
2. Creating Tuples *(single vs multiple elements)*
3. Indexing and Slicing Tuples
4. Tuple Packing and Unpacking
5. Basic Tuple Operations

**Section 2: Practical Usage**

1. Returning Multiple Values from Functions
2. Nested Tuples and Complex Structures
3. Iterating Through Tuples
4. Tuples as Dictionary Keys
5. Performance Advantages Over Lists

**Section 3: Advanced Concepts**

1. Named Tuples with collections Module
2. Immutable Data Design Patterns
3. Tuple Best Practices and Use Cases

**4. Dictionaries – Topics List**

**Section 1: Foundation**

1. Introduction to Dictionaries – Key-Value Stores
2. Creating and Accessing Dictionary Elements
3. Adding and Updating Entries
4. Removing Entries (pop(), popitem(), clear())
5. Dictionary Mutability Explained

**Section 2: Built-in Functions & Operations**

1. Iterating Over Keys, Values, and Items
2. Using get() and setdefault()
3. Merging Dictionaries (update(), unpacking with \*\*)
4. Dictionary Comprehensions
5. Checking for Key Existence (in operator)

**Section 3: Advanced Patterns**

1. Nested Dictionaries and Complex Structures
2. Using defaultdict from collections
3. Counter for Frequency Analysis
4. OrderedDict for Preserving Order *(Python 3.7+ default)*
5. Dynamic Data Handling with Dictionaries

**Section 4: Best Practices**

1. Memory and Performance Considerations
2. Avoiding Common Anti-Patterns
3. Mini Project – Word Frequency Counter

**5. Sets – Topics List**

**Section 1: Foundation**

1. Introduction to Sets – Unique Data Collections
2. Creating Sets *(curly braces vs set())*
3. Adding and Removing Elements (add(), discard(), remove(), clear())
4. Basic Set Operations *(union, intersection, difference)*
5. Checking Membership with in

**Section 2: Advanced Set Operations**

1. Symmetric Difference and Subsets
2. frozenset – Immutable Sets
3. Filtering and Comprehensions with Sets
4. Performance Benefits of Sets for Lookups
5. Using Sets to Remove Duplicates from Lists

**Section 3: Practical Applications**

1. Deduplication in Data Cleaning
2. Fast Membership Testing
3. Working with Large Datasets and Sets
4. Real-World Use Cases *(e.g., tags, permissions)*

**Combined Master Topic Flow**

If you want to teach **all five core data structures together**, the recommended topic order is:

1. **Strings** – Text processing foundation
2. **Lists** – Basic collections and iteration
3. **Tuples** – Immutable structured data
4. **Dictionaries** – Key-value storage and lookups
5. **Sets** – Unique, fast membership handling

**Advanced Python Conditions – Topics List**

**Section 1: Foundation (Beginner to Intermediate)**

1. Introduction & Vision – Why Mastering Conditions Matters
2. Learning Roadmap: Zero → Hero → Expert
3. Basics of if Statement
4. if-else and if-elif-else
5. Nested Conditions – When and How to Avoid Them

**Section 2: Modern & Clean Coding**

1. Ternary Operators (Inline Conditions)
2. Short-Circuiting with and / or
3. Python 3.10+ match-case Statements
4. Guard Clauses for Clean Code
5. Elegant Condition Chaining with all() and any()

**Section 3: Advanced Patterns**

1. Dictionary-Based Decision Making
2. Strategy Pattern for Complex Decision Logic
3. Dynamic Condition Building at Runtime
4. State Machine Approach to Conditions

**Section 4: Performance & Best Practices**

1. Avoiding Repeated Evaluations in Conditions
2. Measuring Conditional Performance (timeit)
3. Common Pythonic Anti-Patterns to Avoid
4. Writing Readable, Scalable Conditional Logic

**Section 5: Wrap-Up**

1. Quiz & Hands-On Challenges
2. Summary and Next Steps

**Advanced Python Loops – Topics List**

**Section 1: Foundation (Beginner to Intermediate)**

1. Introduction to Loops – Why and When to Use Them
2. while Loop – Syntax and Use Cases
3. for Loop – Iterating Over Sequences
4. range() Function for Controlled Iteration
5. Loop Control Statements: break, continue, pass
6. Nested Loops – Structure and Best Practices

**Section 2: Modern & Pythonic Techniques**

1. Looping with enumerate() and zip()
2. List, Dict, and Set Comprehensions *(Pythonic Loops)*
3. Using else with Loops *(Unique Python Feature)*
4. Iterating Over Dictionaries and Sets
5. Unpacking While Looping (Tuple and Multiple Variables)

**Section 3: Advanced Patterns**

1. Generator Expressions and Lazy Iteration
2. Using itertools for Complex Iterations
3. Infinite Loops and Safe Exit Strategies
4. Dynamic Iteration – Building Loops at Runtime
5. Loop Refactoring: Converting Loops to Functions or Map/Filter

**Section 4: Performance & Best Practices**

1. Optimizing Loop Performance *(Big-O Considerations)*
2. Comparing Loop Speed with timeit
3. Avoiding Loop Anti-Patterns
4. Writing Clean, Readable, and Efficient Loops

**Section 5: Wrap-Up**

1. Hands-On Challenges and Loop Refactoring Exercises
2. Summary, Best Practices, and Next Steps

**Python Functions – Topics List**

**Section 1: Foundation (Beginner)**

*Goal: Understand what functions are, why they are used, and how to write simple ones.*

1. **Introduction to Functions**
   * What are functions?
   * Importance of reusability and modularity
   * Built-in vs. user-defined functions
2. **Defining and Calling Functions**
   * def keyword
   * Syntax and basic structure
   * First simple function example
3. **Function Parameters and Arguments**
   * Positional arguments
   * Default arguments
   * Required vs optional arguments
4. **Return Statement and Multiple Return Values**
   * Returning a single value
   * Returning multiple values using tuples
5. **Variable Scope**
   * Local vs Global variables
   * global keyword
   * Why global variables can be dangerous

**Section 2: Intermediate Concepts**

*Goal: Write more flexible, reusable, and cleaner functions.*

1. **Keyword and Named Arguments**
   * Mixing positional and keyword arguments
   * Practical examples in real-world functions
2. **Variable-Length Arguments**
   * \*args for multiple positional arguments
   * \*\*kwargs for multiple keyword arguments
3. **Docstrings and Documentation**
   * Writing clear docstrings
   * help() and \_\_doc\_\_ attribute
   * Importance of documentation for professionals
4. **Nested Functions (Functions Inside Functions)**
   * Why and when to use nested functions
   * Encapsulation and scope control
5. **Functions Returning Functions** *(Higher-Order Functions)*
   * Closures concept introduction
   * Example: Factory function pattern

**Section 3: Modern & Pythonic Function Techniques**

*Goal: Learn modern techniques and functional programming style.*

1. **Anonymous Functions – lambda**
   * Writing small, quick functions
   * Lambda vs def functions
   * Practical use cases (map, filter, sorted)
2. **Map, Filter, and Reduce**
   * Functional programming concepts
   * Applying transformations and filters on data
3. **List Comprehensions vs Map/Filter** *(Comparison)*
4. **Decorators – Adding Functionality Without Changing Code**
   * Basics of decorators
   * Chaining multiple decorators
   * Real-world use cases *(logging, validation, authentication)*

**Section 4: Advanced Patterns**

*Goal: Apply advanced professional-level design concepts.*

1. **Recursion Basics and Use Cases**
   * Understanding recursion
   * Classic examples: factorial, Fibonacci
   * Base case importance
2. **Tail Recursion and Optimization** *(Conceptual)*
3. **Function as First-Class Citizens**
   * Passing functions as arguments
   * Returning functions from functions
4. **Partial Functions with functools**
   * Creating specialized versions of functions
   * Practical example: fixing certain parameters
5. **Caching and Memoization**
   * Improving performance using lru\_cache
6. **Callbacks and Event-Driven Programming**

**Section 5: Performance & Clean Code**

*Goal: Write efficient, maintainable, and professional-grade functions.*

1. **Avoiding Repeated Code with Helper Functions**
2. **Pure Functions and Side Effects** *(Functional Programming Principle)*
3. **Testing Functions with unittest and pytest**
4. **Performance Measurement with timeit**
5. **Refactoring Long Functions into Smaller Units**

**Section 6: Wrap-Up**

*Goal: Reinforce knowledge with practice and future directions.*

1. **Common Function Anti-Patterns to Avoid**
   * Too many arguments
   * Modifying global state
   * Deep nesting
2. **Real-World Hands-On Challenge**
   * Build a mini project using functions only
3. **Summary and Next Steps**
   * Functions → OOP → Modules → Packages

**Python Modules & Packages – Topics List**

**Section 1: Foundation (Zero → Beginner)**

1. **Introduction to Code Organization**
   * Why organizing code matters in real-world projects
   * Scripts vs. Modules vs. Packages
2. **What is a Module?**
   * Creating and importing your first Python module
   * Understanding import, from ... import, and as
3. **Standard Library Modules**
   * Overview of built-in modules (math, random, datetime, etc.)
   * Real-world use cases
4. **Module Search Path**
   * How Python finds modules (sys.path)
   * Understanding relative vs. absolute imports

**Section 2: Building & Using Packages**

1. **What is a Package?**
   * Difference between a module and a package
   * Role of \_\_init\_\_.py in packages
2. **Creating Your First Package**
   * Structuring folders for packages
   * Sub-packages and nested modules
3. **Importing from Packages**
   * Deep imports and \_\_all\_\_ usage
   * Best practices for package imports
4. **Package Namespacing**
   * Absolute imports vs. relative imports
   * Avoiding circular imports

**Section 3: Advanced Package Management**

1. **Python Package Index (PyPI)**
   * Installing external packages using pip
   * Virtual environments for dependency isolation
2. **Creating & Publishing Your Own Package**
   * Writing setup.py or pyproject.toml
   * Adding metadata and documentation
   * Publishing to PyPI
3. **Versioning & Dependency Management**
   * Semantic versioning (MAJOR.MINOR.PATCH)
   * Using requirements.txt

**Section 4: Professional Practices**

1. **Modular Project Structure**
   * Best folder structures for medium to large projects
   * Separation of concerns (core, utils, tests, config)
2. **Handling Configuration & Secrets**
   * Using config.py and .env files
   * python-decouple or dotenv libraries
3. **Testing Modules & Packages**
   * Writing unit tests for modular code
   * Using unittest and pytest
4. **Performance Considerations**
   * Lazy loading modules
   * Minimizing import overhead

**Section 5: Wrap-Up & Hands-On**

1. **Hands-On Practice: Building a Real Package**
   * Example: Weather data package or text utilities package
   * Installing and importing your package
2. **Debugging Imports & Module Issues**
   * Common import errors and how to fix them
3. **Summary & Best Practices**
   * Key takeaways for scalable, maintainable Python code

**File Handling – Persistent Storage Topics**

**Section 1: Introduction & Fundamentals**

1. **Why File Handling?** – Persistent Storage Concepts
2. Types of Storage – Temporary vs Persistent Data
3. Understanding File Paths (Absolute & Relative)
4. Python open() Function – Modes and Basics (r, w, a, b, x)
5. Reading Files: read(), readline(), readlines()
6. Writing Files: write() and writelines()
7. Closing Files and the with Context Manager

**Section 2: Text File Handling**

1. Creating and Reading Text Files (.txt)
2. Appending Data to Files
3. Searching and Replacing Content in Files
4. Counting Words, Lines, and Characters
5. Processing Large Files Efficiently (Streaming)

**Section 3: Working with CSV Files**

1. Introduction to CSV (Comma-Separated Values)
2. Reading CSV with Python’s csv Module
3. Writing CSV Files (Delimited and Custom Delimiters)
4. Using **Pandas** for Advanced CSV Operations *(Preview of Data Analysis)*

**Section 4: JSON Handling**

1. Introduction to JSON – Why It's Popular
2. Reading and Writing JSON using json Module
3. Nested JSON Parsing and Manipulation
4. Real-World Use Case – Saving App Settings or User Data

**Section 5: Binary Files**

1. Working with Binary Files (rb, wb Modes)
2. Storing and Retrieving Images or Non-Text Data
3. Pickle Module – Object Serialization & Deserialization
4. Security Concerns with Pickle Files

**Section 6: File Management Operations**

1. Using os and pathlib for File Operations
   * Check if a file exists
   * Delete, Rename, Copy, Move files
   * Create directories dynamically
2. Handling Multiple Files in a Directory
3. Recursively Searching Files using os.walk() and glob

**Section 7: Error Handling & Best Practices**

1. Handling File Exceptions (FileNotFoundError, PermissionError, etc.)
2. Safe File Operations with try-except-finally
3. Best Practices for Efficient File Handling

**Section 8: Real-World Mini Projects**

1. Build a Log Analyzer (Read & Process Log Files)
2. Create a CSV-Based Mini Inventory Management System
3. JSON-Based Configuration Manager for an Application
4. Pickle-Based Data Persistence Example

**Advanced Python Exception Handling – Topics List**

**Section 1: Foundation (Beginner)**

1. **Introduction to Exceptions**
   * What are exceptions?
   * Difference between errors and exceptions
   * Why robust error handling matters
2. **Common Built-in Exceptions**
   * ValueError, TypeError, KeyError, etc.
   * Understanding exception hierarchy
3. **Basic try-except Usage**
   * Catching errors gracefully
   * Writing your first error-handling block

**Section 2: Intermediate Techniques**

1. **Using else and finally Blocks**
   * When to use else
   * Resource cleanup with finally
2. **Catching Multiple Exceptions**
   * Using tuples in except
   * Best practices for handling multiple error types
3. **Exception Chaining (raise from)**
   * Preserving original error context
   * Debugging benefits
4. **Logging Exceptions**
   * Using logging module
   * Storing detailed error reports

**Section 3: Advanced Concepts**

1. **Custom Exception Classes**
   * Creating your own exception hierarchy
   * Adding meaningful error messages
2. **Raising Exceptions Intentionally**
   * Validating input and raising specific exceptions
3. **Context Managers and Exception Safety**
   * with statements for safe resource handling
4. **Assertions vs Exceptions**
   * When to use assert vs raise
5. **Re-raising Exceptions**
   * Passing exceptions up the call stack

**Section 4: Performance & Best Practices**

1. **Performance Considerations of Exceptions**
   * Cost of raising exceptions
   * Avoiding overuse in performance-critical code
2. **Exception Handling Anti-Patterns**
   * Bare except blocks
   * Silent failures
   * Catching everything without action
3. **Graceful Application Shutdown**
   * Handling fatal errors cleanly
   * Example: try-except at top-level main()

**Section 5: Practical Implementation**

1. **Error Handling in Real Projects**
   * File I/O operations
   * Network/API calls
   * Database interactions
2. **Debugging Techniques**
   * Using traceback module
   * Capturing stack traces
3. **Testing Error Handling**
   * Unit testing exceptions with pytest

**Section 6: Wrap-Up**

1. **Hands-On Exercises**
   * Real-world exception handling challenges
2. **Summary & Best Practices Recap**

**OOP in Python – Topics List**

**Section 1: Introduction to OOP (Foundation)**

1. What is OOP? – Real-World Examples & Benefits
2. Key OOP Concepts: **Encapsulation, Abstraction, Inheritance, Polymorphism**
3. Difference Between **Procedural vs OOP Programming**
4. Python and OOP – Why Python is Different (Dynamic & Flexible OOP)

**Section 2: Classes & Objects (Core Building Blocks)**

1. Creating Your First Class and Object
2. \_\_init\_\_ Constructor Method and self
3. Instance Variables vs Class Variables
4. Instance Methods vs Class Methods vs Static Methods
5. \_\_str\_\_ and \_\_repr\_\_ for Object Representation

**Section 3: Encapsulation & Data Hiding**

1. Public, Protected, and Private Attributes (\_ and \_\_)
2. Getters, Setters, and Property Decorators (@property)
3. Encapsulation Best Practices in Python

**Section 4: Inheritance (Code Reuse)**

1. Introduction to Inheritance – super() and Parent Classes
2. Method Overriding and Extension
3. Multiple Inheritance and MRO (Method Resolution Order)
4. Mixins – Lightweight Code Reuse Pattern

**Section 5: Polymorphism & Abstraction**

1. Method Overloading vs Method Overriding in Python
2. Duck Typing – Pythonic Polymorphism
3. Abstract Base Classes (abc Module)
4. Interfaces and Design Principles

**Section 6: Special & Magic Methods**

1. Understanding Dunder Methods (\_\_add\_\_, \_\_len\_\_, \_\_eq\_\_, etc.)
2. Operator Overloading – Customizing Object Behavior
3. Custom Iterators using \_\_iter\_\_ and \_\_next\_\_
4. Context Managers – \_\_enter\_\_ and \_\_exit\_\_

**Section 7: Advanced OOP Patterns**

1. Composition vs Inheritance – When to Use Each
2. Data Classes (@dataclass in Python 3.7+)
3. Singleton, Factory, and Strategy Design Patterns
4. Dependency Injection and Loose Coupling

**Section 8: Organizing & Scaling OOP Code**

1. Creating Packages and Modules for OOP Projects
2. Namespaces and Scope in OOP
3. Structuring Large OOP Projects (Best Practices)
4. Documentation and Docstrings for Classes

**Section 9: Performance & Best Practices**

1. OOP vs Functional Trade-offs in Python
2. Optimizing OOP Performance
3. Common OOP Anti-Patterns to Avoid
4. Writing Pythonic, Scalable OOP Code

**Section 10: Wrap-Up & Capstone**

1. Final OOP Mini-Project – Real-World Problem
2. Refactoring Existing Code into OOP Design
3. Review Quiz & Hands-On Challenge
4. Summary and Roadmap to Design Patterns

**Advanced Python Concepts – Modern Patterns**

**Section 1: Deep Dive into Functions & Functional Programming**

1. Recursion – Writing Elegant Recursive Functions
2. Anonymous Functions (lambda) in Depth
3. map(), filter(), reduce() – Functional Style
4. Higher-Order Functions *(Functions as First-Class Citizens)*
5. Closures – Capturing State in Functions
6. Decorators – Writing and Applying Custom Decorators
7. Built-in Decorators: @staticmethod, @classmethod, @property

**Section 2: Iterators and Generators**

1. Understanding Iterators and the Iterator Protocol
2. Building Custom Iterators with \_\_iter\_\_ and \_\_next\_\_
3. Generators – yield and yield from
4. Generator Pipelines for Big Data Processing
5. Memory Efficiency: Lazy Evaluation with Generators

**Section 3: Comprehensions & Modern Collections**

1. Advanced List Comprehensions – Nested and Conditional
2. Dictionary and Set Comprehensions
3. Using collections Module:
   * Counter
   * defaultdict
   * namedtuple
   * deque
   * OrderedDict
4. ChainMap and Other Specialized Collections

**Section 4: Error Handling & Robust Code**

1. Custom Exceptions and Exception Hierarchies
2. Context Managers and with Statement
3. Writing Your Own Context Managers (\_\_enter\_\_, \_\_exit\_\_)
4. Resource Management and Clean Code Principles

**Section 5: Patterns & Modern Techniques**

1. Unpacking and Multiple Assignment Patterns
2. Structural Pattern Matching (match-case) – Python 3.10+
3. Strategy Pattern Implementation in Python
4. Factory Pattern – Dynamic Object Creation
5. Observer Pattern for Event-Driven Programming
6. State Machine Approach to Complex Logic

**Section 6: Performance, Debugging & Best Practices**

1. Profiling Code (cProfile, timeit)
2. Writing Memory-Efficient Code
3. Pythonic Refactoring Techniques
4. Debugging with pdb and breakpoint()
5. Code Quality and Linting (flake8, black, pylint)

**Section 7: Wrap-Up & Projects**

1. Hands-On: Build a Custom Decorator Library
2. Implementing a Mini Framework Using Generators and Decorators
3. Best Practices Recap – Writing Clean, Modern Python
4. Final Quiz and Roadmap to Expert-Level Python

**Data Handling & Libraries – Real-World Data Manipulation**

**Section 1: Introduction to Data Handling**

1. Why Data Handling Matters in Modern Applications
2. Understanding Data Formats: CSV, JSON, XML, Excel, SQL
3. Working with Files: Reading & Writing Text and Binary Files
4. Handling Large Data Files Efficiently (Memory Management Basics)

**Section 2: Python Built-in Modules for Data**

1. os and pathlib – File and Directory Management
2. csv Module – Reading and Writing CSV Files
3. json Module – Parsing and Generating JSON
4. pickle – Object Serialization and Deserialization

**Section 3: Core Data Libraries**

1. Introduction to NumPy – Arrays, Vectorization, and Broadcasting
2. Data Cleaning and Transformation with pandas
3. Data Selection, Indexing, and Filtering Techniques in pandas
4. Aggregations and Grouping – Summaries and Reports
5. Handling Missing Data and Data Types

**Section 4: Real-World Data Manipulation**

1. Importing Data from APIs (REST/HTTP Requests)
2. Web Scraping Basics with requests and BeautifulSoup
3. Exporting Data: CSV, JSON, Excel, SQL
4. Combining and Merging Multiple Datasets

**Section 5: Advanced Techniques**

1. Time Series Data Handling with pandas
2. Using pandas with Databases (SQLite, PostgreSQL)
3. Efficient Large-Scale Data Processing (Dask, Vaex)
4. Introduction to Data Visualization (matplotlib & seaborn)

**Section 6: Performance & Best Practices**

1. Vectorization vs. Loops – Writing Faster Data Pipelines
2. Memory Optimization Techniques
3. Data Cleaning Automation – Building Reusable Functions
4. Best Practices for Reproducible Data Workflows

**Section 7: Hands-On & Wrap-Up**

1. End-to-End Mini Project – From Raw Data to Clean Dataset
2. Summary, Resources, and Next Steps (Machine Learning Prep)

**Mini Projects – Consolidating Python Core Skills**

**Section 1: Beginner Level – Build Confidence**

1. **Calculator Program**
   * Basic arithmetic operations with functions
   * Use of loops and conditions for continuous operations
2. **Number Guessing Game**
   * Random number generation
   * Conditions and loops for gameplay
   * User input validation
3. **Password Strength Checker**
   * String operations for checking complexity
   * Conditions to validate strength levels (weak, medium, strong)
4. **Word Counter**
   * Counting words, characters, and special symbols in text
   * String splitting and dictionary usage

**Section 2: Intermediate Level – Combine Multiple Concepts**

1. **Contact Book Application**
   * CRUD operations using **dictionary**
   * Menu-driven program with loops and functions
   * Input validation and search feature
2. **Student Management System**
   * Use of list and dictionary together
   * Store and display student details
   * Calculate grades and percentages
3. **Expense Tracker**
   * Categorize expenses using sets and dictionaries
   * Generate summary reports
   * Save and load data from a text/CSV file *(optional)*
4. **Library Management Mini-System**
   * Track books, borrowers, and return dates
   * Nested dictionaries for complex data structures
   * Conditional logic for late fees

**Section 3: Advanced Level – Real-World Simulation**

1. **Simple Banking Application**
   * Functions for deposit, withdrawal, and balance check
   * Conditions for overdraft protection
   * Use of loops for continuous operations
2. **Quiz Application**
   * Dictionary for question bank
   * Score tracking and performance summary
   * Random question selection
3. **Inventory Management System**
   * Combine lists, tuples, and dictionaries
   * Generate reports (stock in/out)
   * Data persistence *(optional file handling)*
4. **To-Do List CLI App**
   * Add, edit, delete, mark completed tasks
   * Use of sets for completed vs. pending tasks
   * Save to file for session persistence

**Section 4: Capstone Projects**

1. **Data Cleaning Script** *(Practical Real-World Task)*
   * Remove duplicates from lists/sets
   * Clean strings by trimming and normalizing text
   * Standardize data using functions
2. **Text-Based Adventure Game** *(Fun & Interactive)*
   * Nested conditions for decision-making
   * Dictionaries for game states and transitions
   * Loops for continuous play
3. **Mini E-Commerce Simulation** *(Professional Level)*
   * Product catalog using dictionary
   * Shopping cart using lists and sets
   * Functions for checkout and receipt generation

**Section 5: Wrap-Up**

1. Best Practices in Mini Project Development
2. Writing Modular Code with Functions
3. Planning for Larger, Real-World Projects
4. Version Control Basics with Git *(Optional)*
5. Showcase & Presentation of Completed Projects