

Python Programming Syllabus

A Comprehensive Track from Foundations to Advanced Architectures

Level:	Beginner to Advanced	Duration:	8 Weeks (Accelerated / Flexible)
Focus:	Core Language, OOP, & Data Engineering	Format:	Theoretical Concepts & Practical Labs

Module 1: Python Essentials & Environment Setup

- **Introduction to Python:** Core features, execution model, compilation vs. interpretation, and use cases.
- **Environment Configuration:** Setting up Anaconda distribution, working with Jupyter Notebooks, and configuring VS Code.
- **Core Syntax & Variables:** Dynamic typing concepts, explicit memory reference assignment, variable naming conventions, and PEP 8 style guidelines.
- **Primitive Data Types:** Operational workflows with Strings, Integers, Floats, and Booleans.
- **Type Casting & I/O:** Safe type conversions, handling terminal inputs via `input()`, and building clean output presentation using `f-strings`.

Module 2: Control Flow & Advanced Data Structures

- **Conditional Logic:** Structural evaluation using `if`, `elif`, and `else` chains paired with logical operators (`and`, `or`, `not`).
- **Looping Mechanisms:** Definitive `for` loops, conditional `while` loops, nesting layers, and execution modifications using `break`, `continue`, and `pass`.
- **Built-in Sequences:**
 - **Lists:** Zero-indexing mechanics, slice notation matrices, list comprehensions, and mutability operations.
 - **Tuples:** Immutability protections, structural data packing, and variable unpacking methodologies.
- **Mapping & Set Types:**
 - **Dictionaries:** Key-value paired mapping, hash table lookup speeds, structural dictionary comprehensions, and nested manipulation.
 - **Sets:** Element uniqueness enforcement, membership testing, and set algebraic operations (union, intersection, difference).

Module 3: Functional Programming & Modular Architecture

- **Functional Design:** Explicit function definition syntax, parameter declarations, argument parsing mechanisms (positional, keyword, `*args`, and `**kwargs`), and modular return patterns.
- **Scope Resolution:** The LEGB rule (Local, Enclosing, Global, Built-in namespaces) and utilizing variable scopes cleanly.
- **Lambda Expressions:** Designing short anonymous functions and mapping logic natively via `map()`, `filter()`, and `reduce()` tools.
- **Modular Ecosystems:** Code segmentation across physical scripts, packages management, and leveraging core standard libraries (`math`, `datetime`, `os`, `sys`).

Module 4: File Infrastructure & Exception Architecture

- **File Input/Output Operations:** Processing data streaming pathways for structured and unstructured formats (`.txt`, `.csv`, `.json`) using context managers (`with` block).
- **Robust Error Handling:** Multi-tiered safety layers built via `try`, `except`, `else`, and `finally` blocks.
- **Custom Exception Design:** Programmatically raising targeted exceptions and engineering custom domain-specific exception classes.

Module 5: Object-Oriented Programming (OOP)

- **Object Blueprints:** Architectural design using Classes, instantiation properties, and constructor definitions via `__init__`.
- **Attributes & Methods:** Differentiating between Instance methods, Class methods (using `@classmethod`), and Static helpers (using `@staticmethod`).
- **The Four Pillars of OOP:**
 - **Encapsulation:** Access control implementation using `public`, `protected`, and `private` namespace decorations (name mangling protocols).
 - **Inheritance:** Reusability patterns including single, multiple, and multi-level hierarchy structures; explicit parent class mapping via `super()`.
 - **Polymorphism:** Method overriding interfaces and loose duck-typing execution workflows.
 - **Abstraction:** Defining concrete interface standards using Abstract Base Classes (via the `abc` standard module).

Module 6: Advanced Paradigms & Data Engineering Foundations

- **Advanced Iterables:** State retention streaming using custom Iterators, Generators, and lazy evaluation via the `yield` syntax.

- **Decorators & Closures:** Meta-programming patterns, nested closure scopes, and extending functional definitions cleanly.
- **Introduction to Structured Engineering Data:**
 - **NumPy:** Fast vectorized computing matrices, N-dimensional array manipulations, and structural element operations.
 - **Pandas:** Relational DataFrame architectures, missing value resolution vectors, and structural profiling routines.