Python Training Syllabus

Training Overview

Target Audience:

- Technical Engineers / Scientists (ISRO/URSC)

IDE & OS:

- Spyder (Anaconda) , Jupyter Notebook, Terminal/Bash/Windows Prompt/Powershell
- Microsoft Windows / Linux / MacOS

Format:

- Instructor-led, Hands-On Sessions

Requirements:

- System with Any listed OS Above
- Anaconda Suite (Spyder, Jupyter Notebook) Installed
- Necessary Permissions to install Packages
- Terminal/Console/Prompt Access
- Internet Connectivity for PyPI Packages
- Access to Github (Source Code Management)
- VISA supported Equipments (If Any)
    - NI-VISA Supported Equipments
        - Function Generators
        - Oscilloscopes
        - Digital Multimeters
        - Data Acquisition Systems
    - RaspberryPi or Equivalent for Simulation

Date:

- 23rd June 2025 to 27th June 2025 (5 Days)

Timing:

- Session 01 : 09:00 AM - 10:30 AM
- Session 02 : 10:45 AM - 12:30 PM
- Lunch : 12:30 PM - 01:30 PM
- Session 03 : 01:30 PM - 03:00 PM
- Session 04 : 03:15 PM - 05:00 PM

Table of Contents:

Day 1 – Python Foundations: Data Types, Control Flow, Best Practices

Session 1: Python Overview & Coding Practices

- Role of Python in Scientific Computing
- Spyder IDE: Environment, Panels, Console
- Python syntax recap (PEP8, dynamic typing)
- Mutable vs Immutable types
- Type hints (typing module) and annotations

Session 2: Core Data Types & Operations

- Working with environments (venv vs conda)
- Strings (f-strings, formatting, slicing)
- Lists, Tuples, Sets, Dictionaries – use-cases
- Comprehensions with conditions
- Practical lab: Data normalization and encoding

Session 3: Control Flow in Engineering Context

- if, if-else, if-elif-else with real-world decision models
- for, while, break, continue, else
- Pattern generation and conditional logic for sensors

Session 4: Labs and Application Logic

- Lab Exercises:
    - Resistance calculation with conditional logic
    - Classification based on sensor ranges
    - Lookup structures using dict

Day 2 – Functions, Closures, Decorators, Modules

Session 1: Functions Deep Dive

- Defining, calling, returning values
- Positional vs keyword vs default arguments
- *args and **kwargs for flexible APIs
- Docstrings, annotations, type hints
- Writing modular functions
- Functions as data — `map`, `filter`, `reduce`, `functools.partial`

Session 2: Advanced Function Concepts

- Closures – capturing local context
- First-class functions
- Decorators: writing and applying
    - Logging, timing, input validation use-cases
    - `@wraps`, class-based decorators
- `functools`: `lru_cache`, `singledispatch`

Session 3: Organizing Code & Reusability

- Module hierarchy, `__init__.py`, relative imports
- `__name__ == '__main__'` idiom
- Writing and importing modules
- Local vs pip-installed modules
- Build and publish your own `.whl` package
- Python standard library essentials (math, os, sys, pathlib)
- Structuring projects

Session 4: Hands-On Labs

- Real-world examples:
    - Decorator to time sensor calculations
    - Closure for adaptive filter config
    - Modular temperature logger

Day 3 – OOP, Exceptions, File I/O, Testing

Session 1: Object-Oriented Programming Essentials

- Class, object, `__init__`, attributes
- Instance vs class variables
- Inheritance, method overriding

Session 2: Advanced OOP + Pythonic Design

- `@staticmethod`, `@classmethod`, `@property`
- Encapsulation & abstraction
- Dunder methods: `__str__`, `__repr__`, `__eq__`

Session 3: Exception Handling & Robust Code

- `try`, `except`, `else`, `finally`
- Built-in exceptions and raising custom exceptions
- Assertion-based debugging
- Logging vs exception throwing
- Writing custom exception classes

Session 4: File I/O & Unit Testing

- Reading/writing: Text, CSV, JSON, Pickle
- Context managers
- `pathlib` vs `os.path`
- `unittest` for testing scientific functions
- `pytest`, fixtures
- Mocking file and serial input
- Debugging with `pdb`, tracebacks

Day 4 – Scientific Computing with NumPy, Pandas, Matplotlib

Session 1: NumPy for Numerical Computation

- Array creation, slicing, broadcasting
- ndarray internals: shape, strides, dtype
- Vectorized math, dot product, stats
- Vectorized operations, ufuncs, axis logic
- Memory layout, performance benefits
- Engineering simulations with matrices
- Linear algebra: dot, inverse, eig, solve

Session 2: Pandas for Structured Data

- DataFrame creation, selection, filtering
- Aggregation, groupby, joins
- Time series handling, resampling
- Handling missing data, categorical data
- GroupBy patterns and window operations
- Clean, merge, validate telemetry streams

Session 3: Matplotlib for Visualization

- Line, bar, scatter, histograms
- Customizing plots: labels, grids, styles
- Subplots, exporting plots
- Visualization for reports (sensor trends)
- Prospect to Seaborn

Session 4: Labs and Project

- Mini project:
    - Load CSV → Analyze in Pandas → Visualize with Matplotlib
    - Dashboard for telemetry data
    - Prospect to Streamlit

Day 5 – Engineering Interfaces: APIs, Sockets, VISA, UART

Session 1: API Programming with Python

- REST API overview
- REST principles: stateless, JSON, verbs
- Using requests: GET, POST, headers
- `requests`: headers, auth, session pooling
- JSON parsing
- Calling open APIs (e.g., weather, satellite data)
- Designing lightweight data pullers

Session 2: Socket Programming

- TCP/IP overview
- Creating client-server using socket
- Structured message exchange
- Protocol design (binary/JSON)
- Simulated telemetry transfer

Session 3: VISA Instrument Communication

- Introduction to VISA and pyVISA
- Interfacing with instruments: USB/GPIB/RS-232
- SCPI command structure
- SCPI command parsing and read/write
- Reading data, setting configs

Session 4: UART with pySerial

- Basics of Serial (COM) communication
- `pyserial` for UART
- Setup: baud rate, parity, timeout
- Reading/writing data packets
- Syncing device reads with file logging
- Simulating microcontroller communication
- Project: API + Socket + UART unified script for mock data pipeline

Training Outcomes

By the end of this 5-day advanced Python training program, participants will be able to:

1. **Master Python for Scientific and Engineering Applications**
   - Understand Python's internal architecture, memory handling, and performance considerations
   - Write clean, modular, reusable, and well-documented Python code
2. **Apply Advanced Programming Constructs**
   - Use advanced data structures (collections, nested containers) and comprehensions effectively
   - Implement closures and decorators to enhance functionality in a clean, Pythonic way
3. **Design Object-Oriented and Robust Python Applications**
   - Build reusable, testable classes and modules using OOP principles
   - Implement structured exception handling and unit testing for critical systems
   - Organize projects with modules, packages, and version control readiness
4. **Perform Data Analysis and Visualization**
   - Efficiently manipulate large numeric datasets using NumPy and Pandas
   - Analyze telemetry, sensor, and engineering data using structured workflows
   - Create publication-ready visualizations using Matplotlib with full customization
5. **Interface Python with Real-World Systems**
   - Integrate Python with external systems via REST APIs and TCP/IP socket interfaces
   - Communicate with lab instruments using VISA (SCPI) and embedded systems via UART
   - Build end-to-end data pipelines from sensors to analytics dashboard
6. **Develop Hands-On Problem Solving with Mini Projects**
   - Complete multiple context-relevant mini projects: telemetry analyzer, logger, sensor data visualizer, and UART/API interfaces
   - Gain confidence in applying Python to real-world engineering and automation challenges

Deliverables

- ☐ **PDF Materials**
- ☐ **All Notebooks, Scripts, Lab Code**
- ☐ **Mini Projects for each Domain**