Deep Dive — Week 1: Professional Python Foundations

Elite Bootcamp 2025–2026 · System Automation Engineer Path

Setup Instructions (Debian 12)

1. Update system and install Python 3 environment: **sudo apt update && sudo apt install python3 python3-venv python3-pip -y**
2. Create and activate a virtual environment:
python3 -m venv ~/bootcamp_env && source ~/bootcamp_env/bin/activate
3. Install essential tools:
pip install pytest tqdm psutil
4. Create a project workspace:
mkdir -p ~/EliteBootcamp/Week1 && cd ~/EliteBootcamp/Week1

Section 1 — Syntax, Functions, and Modules

Revisit Python fundamentals such as variables, loops, and functions. Focus on clean, readable code that uses type hints and modular design.

Example:

```
def greet_user(name: str) -> None: print(f"Hello, {name}!") if __name__ == "__main__": for user in ["Alice", "Bob", "Charlie"]: greet_user(user)
```

■ Deep Dive Task: Write a function that computes system uptime in hours using psutil.

Section 2 — Object-Oriented Programming (OOP)

OOP helps structure automation scripts into reusable classes. Learn encapsulation, inheritance, and method overriding. Example:

```
class SystemMonitor: def __init__(self, hostname: str): self.hostname = hostname def status(self) -> str: return f"Monitoring {self.hostname}" class CPUMonitor(SystemMonitor): def status(self) -> str: return f"{self.hostname}: CPU usage normal"
```

■ Engineer Note: In Week 10, this structure will help you wrap C libraries using Python classes.

Section 3 — Exception Handling

Use try/except to handle predictable errors gracefully. Define custom exceptions for clarity.

```
class InvalidCommandError(Exception): pass try: raise InvalidCommandError("Invalid CLI option provided.") except InvalidCommandError as e: print(f"Error: {e}")
```

■ Tip: Always log exceptions instead of printing them directly.

Section 4 — Building CLI Tools with argparse

Argparse allows automation scripts to receive command-line parameters. Example:

```
import argparse parser = argparse.ArgumentParser(description="System Info Tool") parser.add_argument("--cpu", action="store_true", help="Show CPU usage") parser.add_argument("--mem", action="store_true", help="Show memory usage") args = parser.parse_args()
```

Deep Dive Task: Extend this script to print actual stats using psutil.

Section 5 — Logging for Professionals

Logging is critical in automation systems. Always log info, warnings, and errors to files under /var/log or ~/logs.

```
import logging logging.basicConfig( filename='sysinfo.log', level=logging.INFO, format='[%(asctime)s] %(levelname)s - %(message)s' ) logging.info("System monitor started.")
```

Deep Dive Task: Configure rotating file logs using logging.handlers.RotatingFileHandler.

Mini-Project — SysInfo CLI

Goal: Build a command-line tool that displays CPU, memory, and disk usage with clean logging and argparse.

```
Example run: $ python3 sysinfo.py --all [INFO] CPU: 22% | Memory: 48% |
Disk: 37%
```

Deliverables: • sysinfo.py with argparse + logging • GitHub commit message: "Week 1 – Professional Python Foundations (Completed)" • Short README explaining CLI usage