Python for DevOps: From Scripting to Automation - Complete Course Outline

I. Introduction to Python and the DevOps Ecosystem

* **1.1 Why Python for DevOps?**
    * Readability and rapid development.
    * Extensive standard library and rich ecosystem of packages.
    * Cross-platform compatibility.
    * Strong community support.
* **1.2 Overview of DevOps Principles**
    * CI/CD (Continuous Integration/Continuous Deployment).
    * Infrastructure as Code (IaC).
    * Automation of repetitive tasks.
    * Monitoring and Logging.
* **1.3 Setting Up the Development Environment**
    * **IDEs and Code Editors:** PyCharm, VS Code, Jupyter Notebooks.
    * **Python Version Management:** `pyenv`.
    * **Virtual Environments:** `venv` and `virtualenv`.
    * **Dependency Management:** `pip`, `requirements.txt`, `poetry`/`pipenv`.

II. Python Basics (The Foundation)

* **2.1 Syntax and Structure**
    * Indentation, comments, and docstrings.
* **2.2 Variables and Data Types**
    * Primitive and complex types (lists, tuples, sets, dictionaries).
* **2.3 Operators**
    * Arithmetic, comparison, assignment, logical, identity, membership.
* **2.4 Control Flow**
    * Conditional statements (`if`, `elif`, `else`).
    * Looping (`for`, `while`) and loop control (`break`, `continue`, `pass`).
* **2.5 Functions**
    * Defining functions (`def`).

    * Parameters (positional, keyword, default, `*args`, `**kwargs`).
    * Return values and scope (LEGB).
* **2.6 File Handling (Crucial for DevOps)**
    * Reading/writing files, context managers (`with` statement).
    * Common file formats: `.txt`, `.json`, `.yaml`, `.csv`.
* **2.7 Error and Exception Handling (Basics)**
    * `try`, `except`, `else`, `finally` blocks.
    * Raising exceptions with `raise`.

III. Intermediate Python for Automation
* **3.1 Working with the Operating System**
    * The `os`, `sys`, and `pathlib` modules.
* **3.2 Command-Line Execution and Scripting**
    * The `subprocess` module (`subprocess.run()`, `Popen`).
* **3.3 Date and Time Operations**
    * The `datetime` module.
* **3.4 Regular Expressions (regex) - (Text Processing Powerhouse)**
    * The `re` module (`re.search()`, `re.match()`, `re.findall()`, `re.sub()`).
    * Common patterns for logs (IP addresses, timestamps, error codes).
* **3.5 Data Serialization Formats (Config & Data Handling)**
    * JSON (`json` module), YAML (`pyyaml`), INI (`configparser`), environment variables.
* **3.6 Lambda Functions and Functional Tools**
    * Anonymous functions with the `lambda` keyword.
    * Built-in tools: `map()`, `filter()`, `sorted()`.
    * DevOps use cases: quick data transformations and filtering.

IV. Advanced Python Concepts
* **4.1 Advanced Exception Handling**
    * Built-in exception hierarchy.
    * Creating custom exceptions (e.g., `ConfigurationError`).
    * Exception chaining (`raise ... from ...`).

    *   Logging exceptions (`logging.exception()`).
*   **4.2 Type Annotations and Hints**
    *   Purpose: code clarity, maintainability, static analysis.
    *   Annotating variables, parameters, and return types (`str`, `int`, `List`, `Dict`, `Optional`).
    *   Using `mypy` for static type checking.
*   **4.3 Decorators**
    *   Concept: functions that modify other functions.
    *   Syntax using `@`.
    *   Creating custom decorators (e.g., for logging, timing, retry logic).
    *   Built-in decorators: `@staticmethod`, `@classmethod`.

V. Concurrency and Parallelism
*   **5.1 Introduction to Concurrency**
    *   CPU-bound vs. I/O-bound tasks.
    *   The Global Interpreter Lock (GIL).
*   **5.2 Multi-threading for I/O-bound Tasks**
    *   The `threading` module.
    *   The `ThreadPoolExecutor` for managing thread pools.
    *   DevOps use case: parallel status checks for multiple servers.
*   **5.3 Multi-processing for CPU-bound Tasks**
    *   The `multiprocessing` module.
    *   Bypassing the GIL for parallel computation.
    *   DevOps use case: parallel processing of large datasets.

VI. Data Analysis for DevOps (Pandas & NumPy)
*   **6.1 Introduction to NumPy**
    *   Purpose: efficient numerical computations.
    *   Core concept: the `ndarray` object.
    *   DevOps use case: processing numerical monitoring data.
*   **6.2 Introduction to Pandas (The Workhorse)**
    *   Purpose: data manipulation and analysis.

    *   Core structures: `Series` and `DataFrame`.
    *   Data ingestion, inspection, selection, filtering, cleaning.
    *   Grouping and aggregation with `groupby()`.
    *   DevOps use cases: analyzing metrics, parsing billing reports, processing test results.

VII. Object-Oriented Programming (OOP) for Scalable Scripts
*   **7.1 Principles of OOP**
    *   Classes, objects, and the `self` parameter.
*   **7.2 The Four Pillars of OOP**
    *   **Encapsulation:** Bundling data and methods.
    *   **Abstraction:** Hiding complex implementation.
    *   **Inheritance:** Creating child classes.
    *   **Polymorphism:** Unified interface for different forms.
*   **7.3 Special (Magic/Dunder) Methods**
    *   `__init__()`, `__str__()`, `__repr__()`.
*   **7.4 OOP in a DevOps Context**
    *   Modeling infrastructure components (e.g., `Server`, `Database` classes).
    *   Building reusable automation libraries.
    *   Structuring large projects for maintainability.

VIII. Python for Core DevOps Domains
*   **8.1 Working with APIs**
    *   REST API fundamentals.
    *   The `requests` library (`get`, `post`, `put`, `delete`).
    *   Handling authentication (API keys, tokens, OAuth).
    *   Use case: automating interactions with cloud and DevOps tool APIs.
*   **8.2 CI/CD Pipeline Integration**
    *   Writing custom pipeline scripts for Jenkins, GitLab CI, GitHub Actions.
    *   Automating build, test, and deployment stages.

IX. Testing, Logging, and Packaging

*   **9.1 Testing Python Code**
    *   The `unittest` framework.
    *   The `pytest` framework (industry standard).
    *   Writing unit tests for automation scripts.
    *   Mocking external dependencies (`unittest.mock`).
*   **9.2 Logging**
    *   The `logging` module (vs. `print` statements).
    *   Log levels (DEBUG, INFO, WARNING, ERROR, CRITICAL).
    *   Configuring handlers, formatters, and log rotation.
*   **9.3 Packaging and Distributing Scripts**
    *   Project structure (`setup.py`, `pyproject.toml`).
    *   Creating installable packages with `setuptools`.
    *   Creating executable entry-points.