Here's a complete roadmap of **Python topics**, structured from **Beginner** to **Advanced** to **Master/Expert** level:

🟢 Beginner Level

Focus: Syntax, basic data types, control flow

1. **Introduction to Python**
   - Installation (Python, pip)
   - Running Python scripts
   - Python IDEs (VS Code, PyCharm, Jupyter)
2. **Basic Syntax**
   - Comments
   - Indentation
   - Variables and Constants
3. **Data Types**
   - Numbers (int, float, complex)
   - Strings
   - Booleans
   - Type casting
4. **Operators**
   - Arithmetic
   - Comparison
   - Logical
   - Bitwise
   - Assignment
   - Membership & Identity
5. **Control Flow**
   - `if`, `elif`, `else`
   - Loops: `for`, `while`
   - `break`, `continue`, `pass`
6. **Data Structures**
   - Lists
   - Tuples
   - Sets
   - Dictionaries
   - Comprehensions (list, dict, set)
7. **Functions**
   - Defining and calling functions
   - Arguments & return values
   - `*args` and `**kwargs`
   - Lambda functions
   - Scope (local, global)
8. **String Handling**
   - String methods
   - f-strings
   - Formatting and slicing
9. **Basic File I/O**
   - Reading and writing files
   - `with` statement
10. **Exception Handling**
    - `try`, `except`, `finally`
    - `raise`

🟡 Intermediate Level

Focus: OOP, modules, packages, debugging, virtual environments

1. **Object-Oriented Programming (OOP)**
   - Classes and Objects
   - Constructors
   - Inheritance
   - Encapsulation
   - Polymorphism
   - Magic/Dunder methods (`__str__`, `__init__`, etc.)
2. **Modules and Packages**
   - Importing modules
   - Creating custom modules
   - `__init__.py`
   - Standard library usage
3. **Virtual Environments**
   - `venv`, `pipenv`, `virtualenv`
   - `requirements.txt`
4. **Pythonic Features**
   - Enumerate
   - Zip
   - Any, all
   - `map()`, `filter()`, `reduce()`
5. **Decorators and Closures**
   - Function closures
   - Writing decorators
   - `functools.wraps`
6. **Generators and Iterators**
   - `yield` keyword
   - Generator expressions
   - Custom iterators
7. **Error Handling Best Practices**
   - Custom exceptions
   - Logging (`logging` module)
8. **Unit Testing**
   - `unittest`, `pytest`, `assert`
9. **Comprehensions Deep Dive**
   - Nested comprehensions
   - Conditionals in comprehensions
10. **Context Managers**
    - `with` keyword
    - Custom context managers (`__enter__`, `__exit__`)

🔴 Advanced Level

Focus: Advanced Python internals, concurrency, performance, typing

1. **Advanced OOP**
   - Metaclasses
   - Multiple Inheritance
   - MRO (Method Resolution Order)
2. **Functional Programming**
   - `functools`, `itertools`, `operator`
   - Currying, partial functions
3. **Concurrency and Parallelism**
   - Threading
   - Multiprocessing
   - Asyncio (async/await)
   - Event loop, coroutines, tasks
4. **Typing and Annotations**
   - Static typing with `typing`
   - `TypedDict`, `Protocol`, `Generic`
   - `mypy` for type checking
5. **Memory Management**
   - Garbage collection
   - `gc` module
   - `sys.getsizeof()`
6. **Descriptors and Properties**
   - `@property`
   - Getter, setter, deleter
   - Custom descriptors (`__get__`, `__set__`)
7. **Advanced Decorators**
   - Class decorators
   - Parameterized decorators
8. **Data Classes**
   - `@dataclass`
   - `field()`, `asdict()`
9. **Serialization**
   - `pickle`, `json`, `marshal`
10. **Debugging & Profiling**
    - `pdb`, `cProfile`, `line_profiler`

🟣 Master Level

Focus: Deep internals, performance, system programming, best practices

1. **CPython Internals**
   - Bytecode
   - Python execution model
   - Disassembling with `dis`
2. **Memory Optimization**
   - Slots (`__slots__`)
   - Interning
   - Object mutability and memory layout
3. **Custom Imports**
   - Import hooks
   - `importlib` customization
4. **Writing C Extensions**
   - Python C API
   - `ctypes`, `cffi`
5. **JIT and Alternative Runtimes**
   - PyPy
   - Numba
   - Cython
6. **Metaprogramming**
   - Reflection (`getattr`, `setattr`, `hasattr`)
   - Code generation
   - AST manipulation
7. **Build Tools and Packaging**
   - `setuptools`, `poetry`, `pip`
   - Creating PyPI packages
8. **Security**
   - Secure coding practices
   - Avoiding common vulnerabilities (e.g., code injection)
9. **Best Practices and Design Patterns**
   - SOLID principles
   - GoF design patterns in Python
   - Clean Code practices
10. **Domain-Specific Mastery (Optional paths)**
    - Web Development: Django, Flask, FastAPI
    - Data Science: Pandas, NumPy, Matplotlib
    - ML/AI: Scikit-learn, TensorFlow, PyTorch
    - DevOps: Automation scripts, Ansible, Docker + Python
    - Networking: `socket`, `asyncio`, `requests`