As a **3-year Python developer**, you should have a **solid understanding of core and advanced concepts** to handle real-world projects efficiently. Below are key **Python concepts** and **skills** you should master:

**🔹 1. Advanced Python Concepts**

| **Concept** | **Description** |
| --- | --- |
| **Iterators & Generators** | Lazy evaluation, yield, memory-efficient loops |
| **Decorators** | @staticmethod, @property, custom decorators for logging & timing |
| **Context Managers** | with open() as file:, custom \_\_enter\_\_() and \_\_exit\_\_() |
| **Metaclasses** | Modify class creation dynamically (type, \_\_new\_\_) |
| **Descriptors** | \_\_get\_\_, \_\_set\_\_, used in ORM frameworks (Django, SQLAlchemy) |
| **Multithreading & Multiprocessing** | threading.Thread, multiprocessing.Process |
| **Async Programming** | asyncio, await, aiohttp for high-performance I/O tasks |
| **Memory Management** | gc, weakref, deep copy vs shallow copy |
| **Python Internals** | How Python handles objects, memory, GIL, and bytecode |

**🔹 2. Object-Oriented Programming (OOP)**

| **Concept** | **Description** |
| --- | --- |
| **Encapsulation** | Private/public variables, \_var, \_\_var |
| **Inheritance** | Single, multiple, and multilevel inheritance |
| **Polymorphism** | Method overloading, method overriding |
| **Abstract Classes** | ABC, @abstractmethod |
| **Mixin Classes** | Code reuse across unrelated classes |

**🔹 3. Design Patterns in Python**

| **Pattern** | **Usage** |
| --- | --- |
| **Singleton** | Ensure a class has only one instance (used in DB connections) |
| **Factory Pattern** | Object creation without exposing logic |
| **Observer Pattern** | Event-driven programming (e.g., real-time notifications) |
| **Strategy Pattern** | Define a family of algorithms, encapsulate, and make them interchangeable |

**🔹 4. Database and ORMs**

| **Concept** | **Description** |
| --- | --- |
| **SQL & NoSQL** | PostgreSQL, MySQL, MongoDB, Redis |
| **ORMs** | SQLAlchemy, Django ORM |
| **Database Optimization** | Indexing, Query Optimization, Transactions |

**🔹 5. Web Development & APIs**

| **Concept** | **Description** |
| --- | --- |
| **RESTful APIs** | FastAPI, Flask, Django REST Framework |
| **GraphQL** | Query-based APIs (GraphQL with Ariadne, Strawberry) |
| **Authentication & Security** | OAuth2, JWT, CSRF, SQL Injection prevention |
| **WebSockets** | Real-time communication using websockets, socket.io |

**🔹 6. Performance Optimization**

| **Concept** | **Description** |
| --- | --- |
| **Profiling & Debugging** | cProfile, line\_profiler, py-spy |
| **Caching** | functools.lru\_cache, Redis cache |
| **Memory Optimization** | sys.getsizeof(), gc.collect(), weak references |
| **Concurrency** | asyncio vs threading vs multiprocessing |

**🔹 7. DevOps & Deployment**

| **Concept** | **Description** |
| --- | --- |
| **Containerization** | Docker, Docker Compose |
| **CI/CD** | GitHub Actions, Jenkins, GitLab CI |
| **Server Management** | Nginx, Gunicorn, Supervisor for process management |
| **Cloud Services** | AWS (Lambda, S3, EC2), Azure, GCP |
| **Monitoring & Logging** | ELK Stack, Prometheus, Grafana, logging module |

**🔹 8. Unit Testing & Test-Driven Development (TDD)**

| **Concept** | **Description** |
| --- | --- |
| **Testing Frameworks** | pytest, unittest, doctest |
| **Mocking & Patching** | unittest.mock for simulating dependencies |
| **Integration Testing** | Test APIs, database queries |
| **CI/CD Testing** | Automate tests in pipelines |

**🔹 9. Distributed Systems & Messaging Queues**

| **Concept** | **Description** |
| --- | --- |
| **Message Queues** | RabbitMQ, Kafka, Redis Pub/Sub |
| **Task Queues** | Celery for background job processing |
| **Microservices** | FastAPI, Flask-based services with async messaging |
| **Event-Driven Architecture** | Webhooks, Kafka event streaming |

**🔹 10. Machine Learning & AI (If You’re Interested)**

| **Concept** | **Description** |
| --- | --- |
| **ML Libraries** | Scikit-learn, TensorFlow, PyTorch |
| **Data Preprocessing** | Pandas, NumPy, Feature Engineering |
| **Model Deployment** | FastAPI with TensorFlow/PyTorch models |
| **MLOps** | Model monitoring, retraining pipelines |

**💡 Next Steps**

As a **3-year Python developer**, you should:  
✅ **Master advanced Python concepts (iterators, generators, async, OOP)**  
✅ **Learn system design (APIs, databases, caching, scaling strategies)**  
✅ **Optimize performance (profiling, concurrency, memory management)**  
✅ **Understand DevOps & deployment (Docker, CI/CD, cloud platforms)**

Would you like to dive deeper into any **specific topic** from this list? 🚀