Advanced Python Developer Handbook (Extended Edition)

1. Python Execution Model & Memory Management

Example:

```
x = 10
y = x
print(id(x), id(y))
```

- • Python uses reference-based memory management.
- • Everything in Python is an object.
- • Variables store references, not actual values.
- • Garbage collection is automatic (reference counting + cyclic GC).
- • Use id() to check memory reference.

2. Deep Dive into Data Types

Example:

```
import copy
a = [1,2,[3,4]]
b = copy.deepcopy(a)
```

- • Mutable: list, dict, set
- • Immutable: int, float, str, tuple
- • Shallow copy vs Deep copy

3. Advanced Functions

Example Decorator:

```
def decorator(func):
    def wrapper():
        print('Before function')
        func()
    return wrapper
```

- • First-class functions

- • Lambda expressions
- • Closures
- • Decorators

4. Object Oriented Programming Advanced

Example:

```
class Employee:
    company = 'ABC'
    def __init__(self, name):
        self.name = name
```

- • Instance vs Class variables
- • Method types: instance, classmethod, staticmethod
- • Dunder methods (__str__, __repr__)
- • Abstract Base Classes

5. Exception Handling Internals

Example:

```
class CustomError(Exception):
    pass
```

- • Custom exceptions
- • Exception hierarchy
- • finally block usage

6. File Handling & Context Managers

Example:

```
with open('data.txt','r') as f:
    content = f.read()
```

- • with statement ensures automatic resource cleanup
- • Reading JSON, CSV files

7. Modules, Packages & Imports

Example:

```
if __name__ == '__main__':
    print('Executed directly')
```

- • Absolute vs Relative imports
- • __name__ variable
- • Creating reusable modules

8. Virtual Environments & Dependency Management

- • python -m venv venv
- • Activate environment
- • pip freeze > requirements.txt

9. Iterators & Generators

Example:

```
def count():
    yield 1
    yield 2
```

- • __iter__ and __next__ methods
- • yield keyword

10. Concurrency & Parallelism

Example (threading):

```
import threading
```

- • threading vs multiprocessing
- • GIL concept
- • Asyncio basics

11. Data Structures & Algorithm Basics

- • Time Complexity (Big-O)
- • List vs Set lookup time
- • Dictionary hashing
- • Common algorithm patterns

12. Testing & Debugging

Example:

```
import unittest
```

- • unittest basics
- • pytest intro
- • Debugging using pdb

13. Logging & Production Code

- • logging module
- • Log levels: DEBUG, INFO, WARNING, ERROR
- • Writing logs to file

14. Database Connectivity

Example:

```
import sqlite3
conn = sqlite3.connect('test.db')
```

- • sqlite3 module
- • Connecting to MySQL/PostgreSQL

15. Web Development Basics

- • Introduction to Flask & FastAPI
- • REST API basics
- • JSON handling

16. Packaging & Distribution

- • setup.py basics
- • Creating pip-installable packages
- • Versioning

17. Code Quality & Best Practices

- • Follow PEP8
- • Use meaningful variable names
- • Write modular code
- • Use virtual environments
- • Use Git effectively

18. Advanced Interview Questions

- • What is GIL?
- • Explain decorators.
- • What is metaclass?
- • Difference between shallow & deep copy?
- • Explain MRO (Method Resolution Order).