Advanced Data Structures

collections module:

namedtuple – tuple with named fields.

deque – fast appends/pops from both ends.

Counter – counting hashable objects.

defaultdict – dictionary with default values.

heapq – priority queues (min-heaps).

array – memory-efficient arrays.

---

2. Iterators, Generators & Coroutines

- **Iterators** – __iter__() and __next__() methods.
- **Generators** – using yield for lazy evaluation.
- **Generator expressions** – memory-efficient loops.
- **Coroutines** – async generators with await and async def.

---

3. Decorators & Metaprogramming

- **Function decorators** – modify functions without changing code.
- **Class decorators** – modify classes.
- **functools** – @lru_cache, @wraps, partial.
- **Metaclasses** – control class creation.

---

4. Context Managers

- with statement for resource management.
- Custom context managers using __enter__ & __exit__.
- contextlib utilities.

---

5. Object-Oriented Advanced Concepts

- **Multiple Inheritance & MRO (Method Resolution Order)**.
- **Abstract Base Classes** (abc module).

- **Property decorators** – @property, @setter, @deleter.
- **Slots** – __slots__ for memory optimization.
- **Descriptors** – controlling attribute access.

---

6. Functional Programming Tools

- map(), filter(), reduce().
- lambda expressions.
- itertools for infinite iterators, permutations, combinations.
- functools.reduce, operator module.

---

7. Concurrency & Parallelism

- **Threading** – for I/O-bound tasks.
- **Multiprocessing** – for CPU-bound tasks.
- **asyncio** – asynchronous programming.
- **Concurrent.futures** – simple threading/multiprocessing interface.

---

8. Advanced File & Data Handling

- Binary data handling (struct module).
- Memory mapping (mmap module).
- Advanced serialization (pickle, marshal, json).

---

9. Type Hinting & Annotations

- Static typing with typing module.
- Union, Optional, Literal, TypedDict.
- Protocol for structural subtyping.

---

10. Performance Optimization

- Profiling (cProfile, timeit).
- Using NumPy, Cython, or PyPy.

- Efficient loops & avoiding global lookups.

---

11. Advanced Error Handling

- Custom exceptions.
- Exception chaining (raise ... from ...).
- Context-specific exception handling.

---

12. Working with the Python Data Model

- Overloading special methods (__add__, __len__, __getitem__, etc.).
- Making classes iterable.
- Rich comparison methods (__lt__, __eq__).