Python Attribute Access & Descriptors — Questions & Answers (Set 10)

Q1. What is the difference between __getattr__ and __getattribute__?

- __getattribute__(self, name): Called for every attribute access, even existing ones. Must delegate to super().__getattribute__(name) to avoid recursion. Very powerful, but intrusive.
- __getattr__(self, name): Called only when the attribute is not found normally. Acts as a fallback handler for missing attributes, useful for dynamic or virtual attributes.

Rule of thumb: use __getattr__ for fallback defaults, __getattribute__ when you need to intercept every lookup.

Q2. What is the difference between properties and descriptors?

- Properties (property): A built-in, high-level way to manage attribute access with getter/setter/deleter. They are implemented internally as descriptors.
- Descriptors: A lower-level protocol defining __get__, __set__, and __delete__. They provide reusable, general-purpose attribute management logic. Used internally by property, functions, methods, staticmethod, classmethod, etc.

Rule of thumb: use property for simple encapsulated attributes, descriptors for reusable cross-class logic.

Q3. Key differences between __getattr__ vs __getattribute__, and properties vs descriptors

Feature	__getattr__	__getattribute__	Properties	Descriptors
When called	Only if attribute missing	Always, for any access	When accessing a specific attribute defined with property	For any attribute bound to a descriptor object
Use case	Provide fallbacks/dynamic attrs	Global interception of all lookups	Simple per-class managed attributes	Reusable, general attribute control
Complexity	Safer, less intrusive	Risky, can cause recursion	High-level, declarative	Low-level, flexible
Example	Defaults, computed fields	Proxies, logging access	Encapsulation, validation	ORMs, type-checked fields